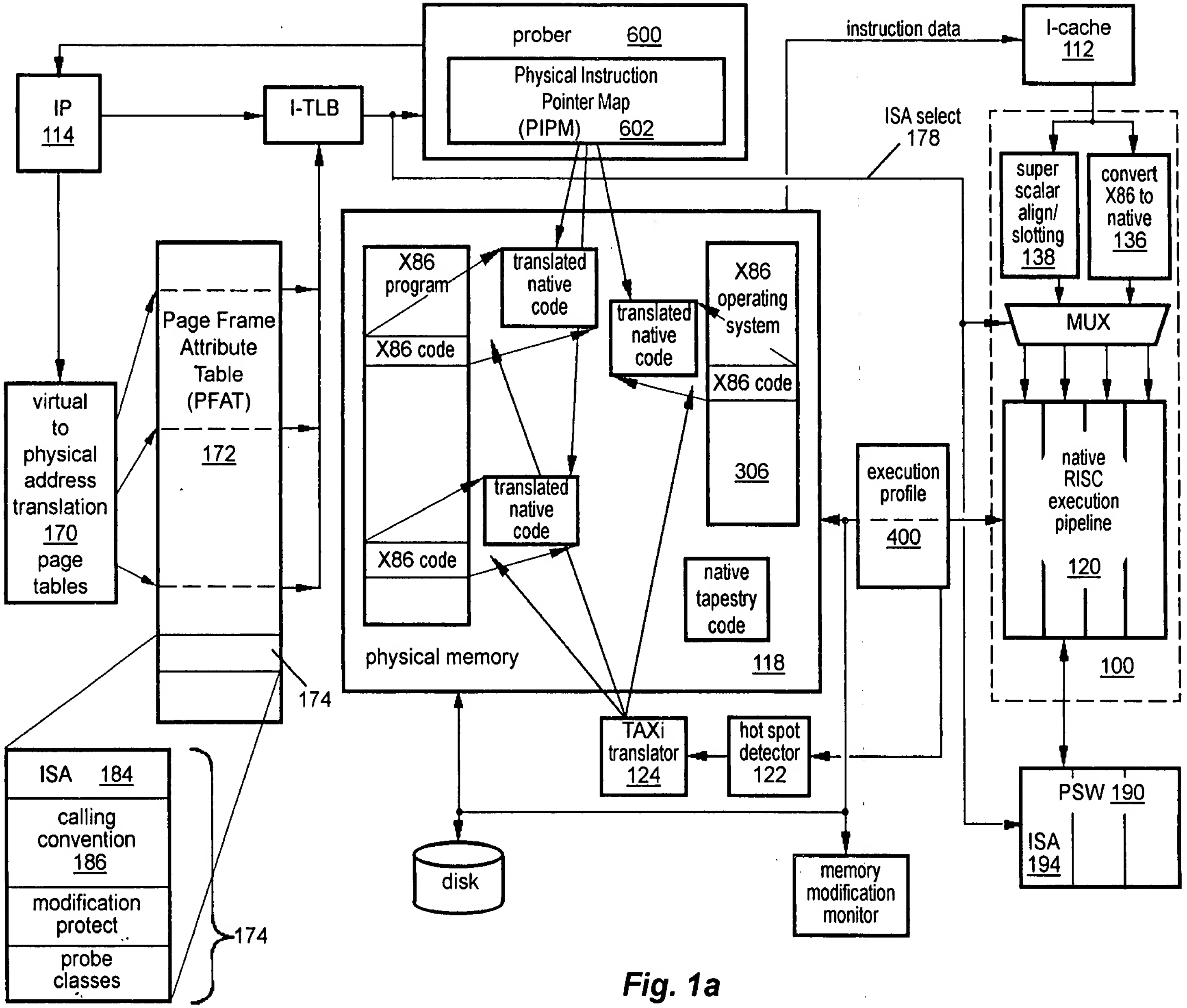

prober 600
Physical Instruction Pointer Map (PIPM) 602
IP 114
I-TLB
instruction data
I-cache 112
ISA select 178
super scalar align/ slotting 138
convert X86 to native 136
MUX
native RISC execution pipeline 120
100
virtual to physical address translation 170 page tables
Page Frame Attribute Table (PFAT) 172
174
X86 program
X86 code
translated native code
translated native code
translated native code
X86 operating system
X86 code
306
native tapestry code
physical memory
118
execution profile 400
TAXi translator 124
hot spot detector 122
disk
memory modification monitor
PSW 190
ISA 194
ISA 184
calling convention 186
modification protect
probe classes
174

Fig. 1a

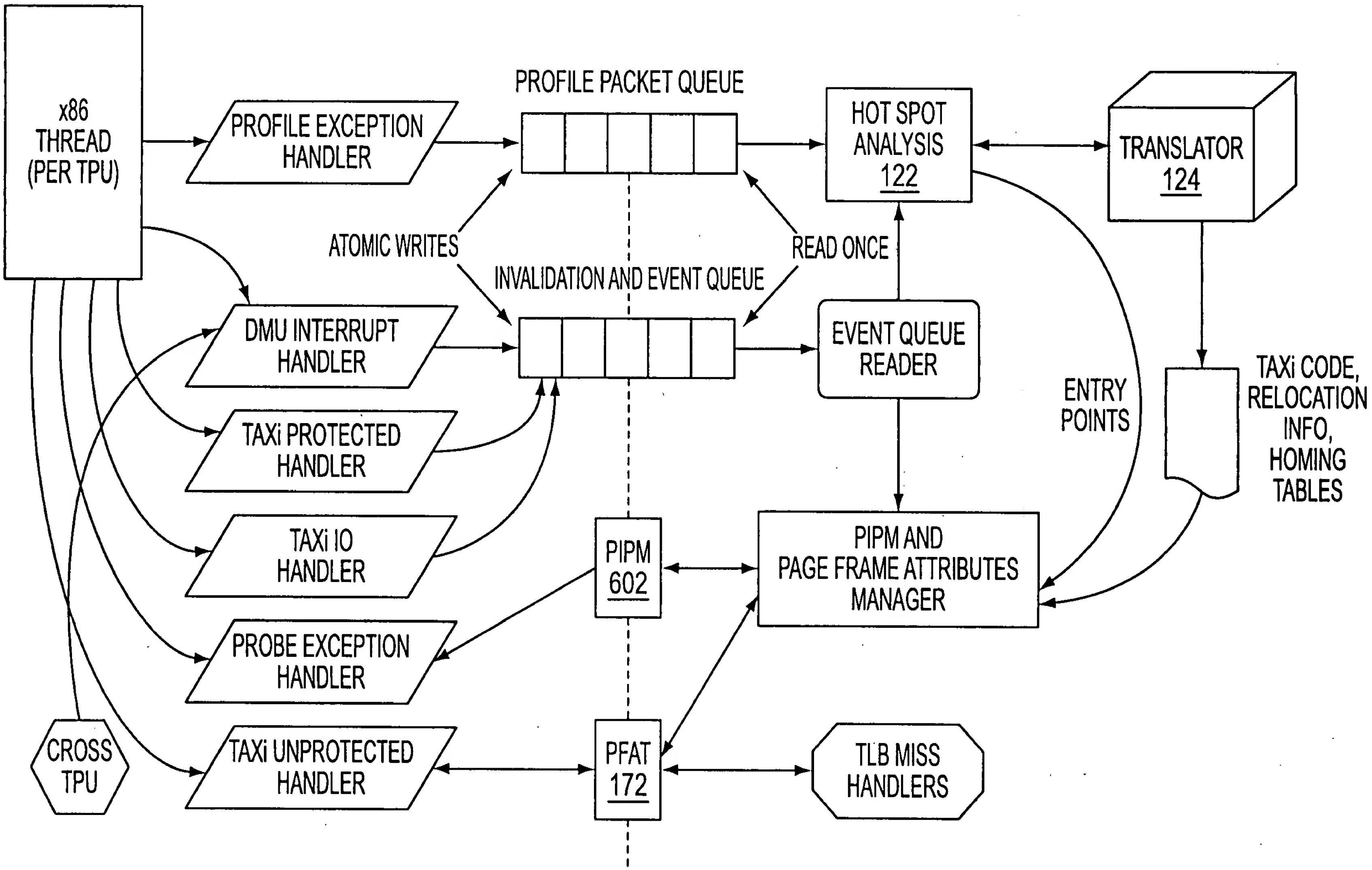
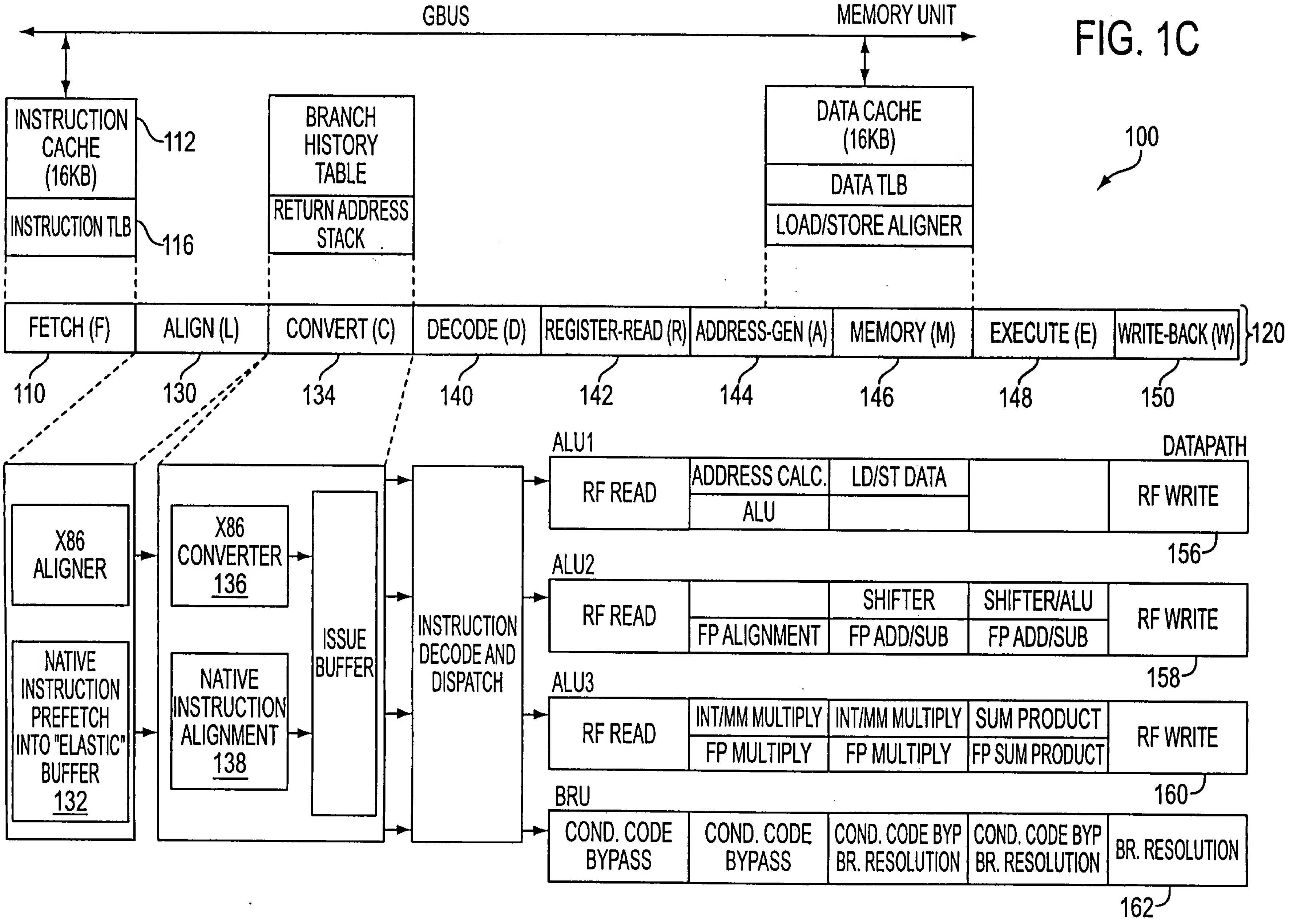

x86
THREAD
(PER TPU)
PROFILE EXCEPTION HANDLER
PROFILE PACKET QUEUE
HOT SPOT ANALYSIS 122
TRANSLATOR 124
ATOMIC WRITES
INVALIDATION AND EVENT QUEUE
READ ONCE
DMU INTERRUPT HANDLER
EVENT QUEUE READER
ENTRY POINTS
TAXi CODE, RELOCATION INFO, HOMING TABLES
TAXi PROTECTED HANDLER
TAXi IO HANDLER
PIPM 602
PIPM AND PAGE FRAME ATTRIBUTES MANAGER
PROBE EXCEPTION HANDLER
CROSS TPU
TAXi UNPROTECTED HANDLER
PFAT 172
TLB MISS HANDLERS

FIG. 1B

FIG. 1C


GBUS
MEMORY UNIT
100
INSTRUCTION CACHE (16KB)
112
INSTRUCTION TLB
116
BRANCH HISTORY TABLE
RETURN ADDRESS STACK
DATA CACHE (16KB)
DATA TLB
LOAD/STORE ALIGNER
FETCH (F)
ALIGN (L)
CONVERT (C)
DECODE (D)
REGISTER-READ (R)
ADDRESS-GEN (A)
MEMORY (M)
EXECUTE (E)
WRITE-BACK (W)
120
110
130
134
140
142
144
146
148
150
X86 ALIGNER
NATIVE INSTRUCTION PREFETCH INTO "ELASTIC" BUFFER 132
X86 CONVERTER 136
NATIVE INSTRUCTION ALIGNMENT 138
ISSUE BUFFER
INSTRUCTION DECODE AND DISPATCH
ALU1
DATAPATH
RF READ
ADDRESS CALC.
ALU
LD/ST DATA
RF WRITE
156
ALU2
RF READ
FP ALIGNMENT
SHIFTER
FP ADD/SUB
SHIFTER/ALU
FP ADD/SUB
RF WRITE
158
ALU3
RF READ
INT/MM MULTIPLY
FP MULTIPLY
INT/MM MULTIPLY
FP MULTIPLY
SUM PRODUCT
FP SUM PRODUCT
RF WRITE
160
BRU
COND. CODE BYPASS
COND. CODE BYPASS
COND. CODE BYP
BR. RESOLUTION
COND. CODE BYP
BR. RESOLUTION
BR. RESOLUTION
162

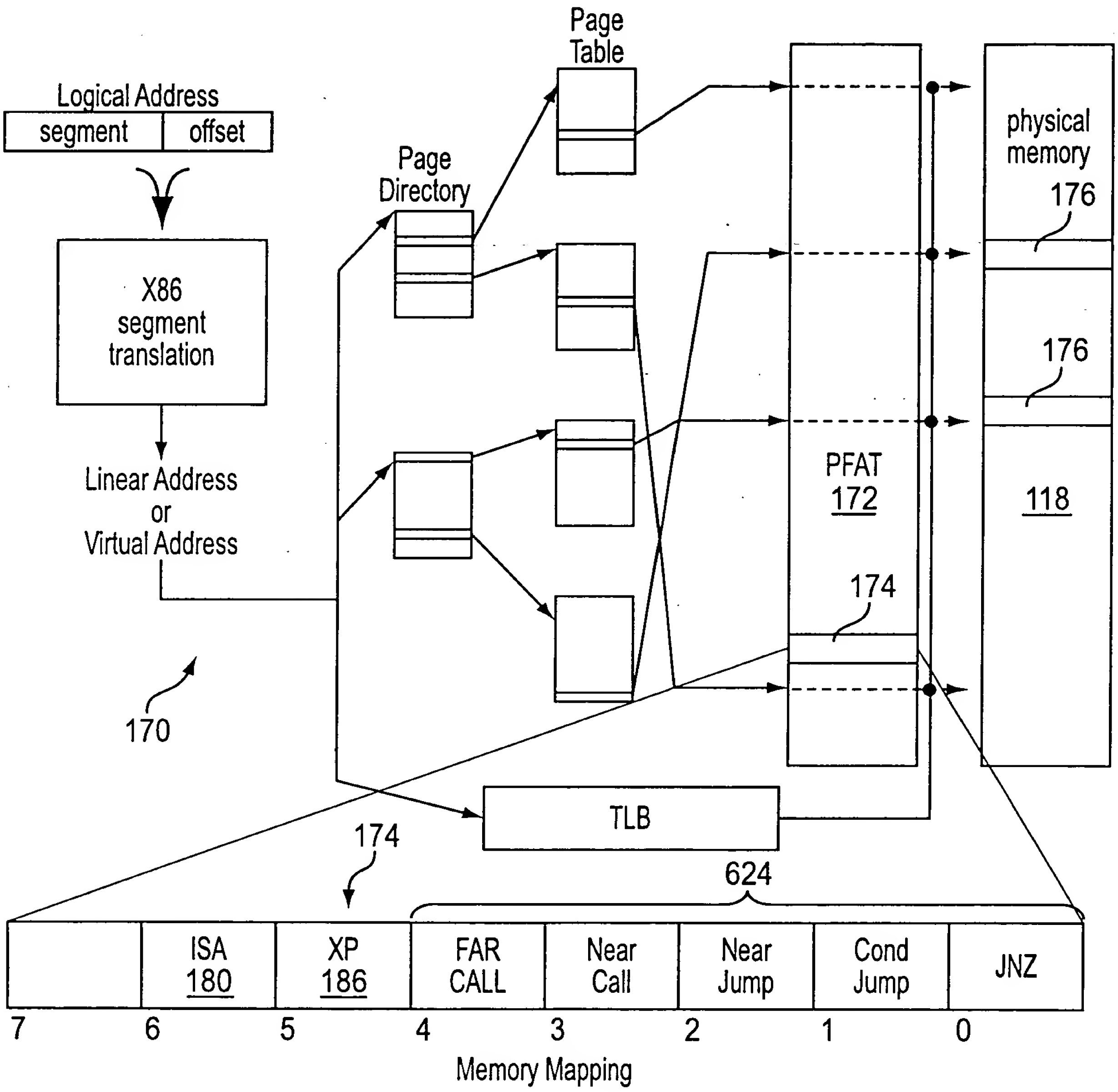

Page Table
Logical Address
segment
offset
Page Directory
X86 segment translation
physical memory
176
176
PFAT
172
118
Linear Address or Virtual Address
174
170
TLB
174
624
ISA 180
XP 186
FAR CALL
Near Call
Near Jump
Cond Jump
JNZ
7
6
5
4
3
2
1
0
Memory Mapping

FIG. 1D

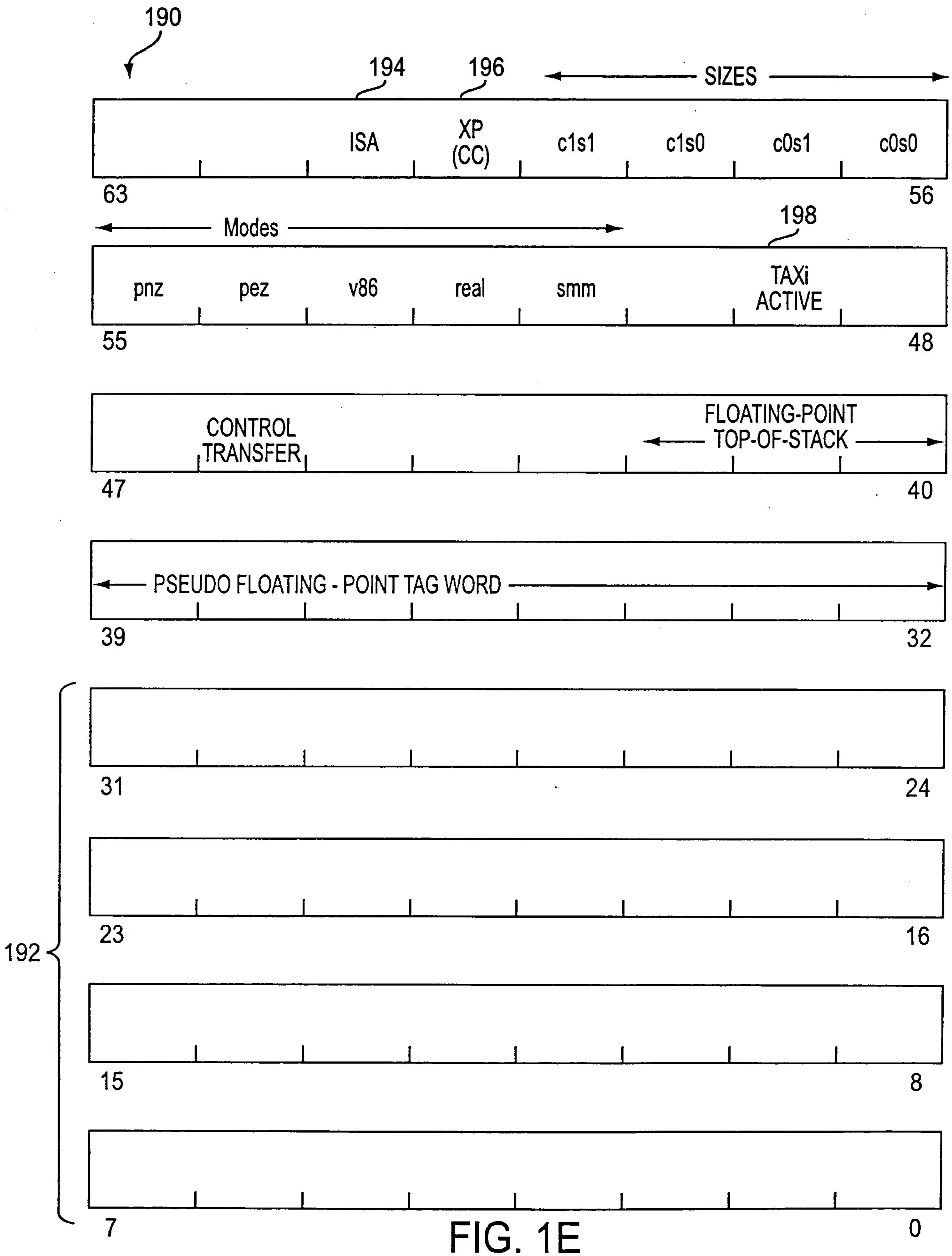

190
194
196
SIZES
ISA
XP (CC)
c1s1
c1s0
c0s1
c0s0
63
56
Modes
198
pnz
pez
v86
real
smm
TAXi ACTIVE
55
48
CONTROL TRANSFER
FLOATING-POINT TOP-OF-STACK
47
40
PSEUDO FLOATING - POINT TAG WORD
39
32
31
24
23
16
192
15
8
7
0

FIG. 1E

I-TLB PROPERTY BITS	DECODED PROPERTY VALUES			INTERPRETATION	INSTRUCTIONS SENT TO:	COLLECT PROFILE TRACE-PACKETS?	PROBE FOR TRANSLATED CODE	I/O MEMORY REFERENCE EXCEPTIONS
	ISA 194	CC 200	PROTECTED					
00	TAP	TAP	NO	NATIVE CODE OBSERVING NATIVE RISCy CALLING CONVENTIONS	NATIVE DECODER	NO	NO	FAULT IF SEG.tio
01	TAP	x86	NO	NATIVE CODE OBSERVING x86 CALLING CONVENTIONS	NATIVE DECODER	NO	NO	FAULT IF SEG.tio
10	x86	x86	NO	x86 CODE, UNPROTECTED - TAX! PROFILE COLLECTION ONLY	x86 HW CONVERTER	IF ENABLED	NO	TRAP IF PROFILING
11	x86	x86	YES	x86 CODE, PROTECTED - TAX! CODE MAY BE AVAILABLE	x86 HW CONVERTER	IF ENABLED	BASED ON I-TLB PROBE ATTRIBUTES	TRAP IF PROFILING

180,182,
184,186

184,186

FIG. 2A

204

	TRANSITION (SOURCE => DEST) ISA & CC PROPERTY VALUES	HANDLER ACTION
212	00 => 00	NO TRANSITION EXCEPTION
214	00 => 01	VECT_xxx_X86_CC EXCEPTION - HANDLER CONVERTS FROM NATIVE TO x86 CONVENTIONS
216	00 => 1x	VECT_xxx_X86_CC EXCEPTION - HANDLER CONVERTS FROM NATIVE x86 CONVENTIONS, SETS UP EXPECTED EMULATOR AND PROFILING STATE
218	01 => 00	VECT_xxx_TAP_CC EXCEPTION - HANDLER CONVERTS FROM x86 TO NATIVE CONVENTIONS
220	01 => 01	NO TRANSITION EXCEPTION
222	01 => 1x	VECT_X86_ISA EXCEPTION [CONDITIONAL BASED ON PCW.X86_ISA_ENABLE FLAG] - SETS UP EXPECTED EMULATOR AND PROFILING STATE
224	1x => 00	VECT_xxx_TAP_CC EXCEPTION - HANDLER CONVERTS FROM x86 TO NATIVE CONVENTIONS
226	1x => 01	VECT_TAP_ISA EXCEPTION [CONDITIONAL BASED PCW.TAP_ISA_ENABLE FLAG] - NO CONVENTION CONVERSION NECESSARY
228	1x => 10	NO TRANSITION EXCEPTION - [PROFILE COMPLETE POSSIBLE, PROBE POSSIBLE]
230	1x => 11	NO TRANSITION EXCEPTION - [PROFILE COMPLETE POSSIBLE, PROBE NOT POSSIBLE]

FIG. 2B

	NAME	DESCRIPTION	TYPE
242	VECT_call_X86_CC	PUSH ARGS, RETURN ADDRESS, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
244	VECT_jump_X86_CC	SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
246	VECT_ret_no_fp_X86_CC	RETURN VALUE TO EAX:EDX, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
248	VECT_ret_fp_X86_CC	RETURN VALUE TO x86 FP STACK, SET UP x86 STATE	FAULT ON TARGET INSTRUCTION
250	VECT_call_TAP_CC	x86 STACK ARGS, RETURN ADDRESS TO REGISTERS	FAULT ON TARGET INSTRUCTION
252	VECT_jump_TAP_CC	x86 STACK ARGS TO REGISTERS	FAULT ON TARGET INSTRUCTION
254	VECT_ret_no_fp_TAP_CC	RETURN VALUE TO RV0	FAULT ON TARGET INSTRUCTION
256	VECT_ret_any_TAP_CC	RETURN TYPE UNKNOWN, SETUP RV0 AND RVDP	FAULT ON TARGET INSTRUCTION

FIG. 2C

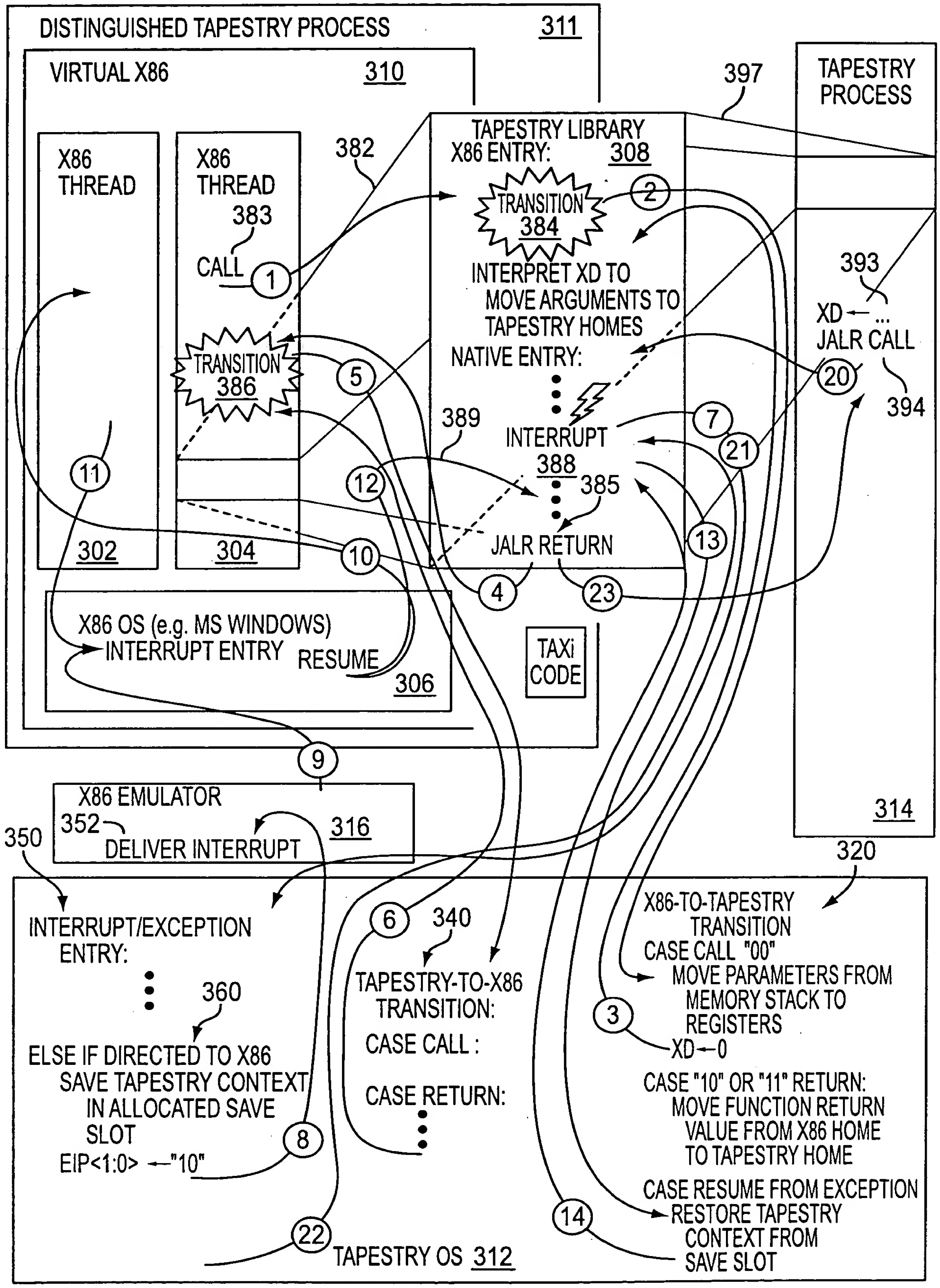

DISTINGUISHED TAPESTRY PROCESS
311
VIRTUAL X86
310
397
TAPESTRY PROCESS
X86 THREAD
X86 THREAD
383
CALL
382
TAPESTRY LIBRARY
X86 ENTRY:
308
TRANSITION
384
INTERPRET XD TO MOVE ARGUMENTS TO TAPESTRY HOMES
NATIVE ENTRY:
TRANSITION
386
393
XD ← ...
JALR CALL
394
389
INTERRUPT
388
385
JALR RETURN
302
304
1
2
3
4
5
6
7
8
9
10
11
12
13
14
20
21
22
23
X86 OS (e.g. MS WINDOWS)
INTERRUPT ENTRY
RESUME
306
TAXi CODE
314
X86 EMULATOR
352
DELIVER INTERRUPT
316
350
320
INTERRUPT/EXCEPTION ENTRY:
360
ELSE IF DIRECTED TO X86
SAVE TAPESTRY CONTEXT IN ALLOCATED SAVE SLOT
EIP<1:0> ←"10"
340
TAPESTRY-TO-X86 TRANSITION:
CASE CALL :
CASE RETURN:
X86-TO-TAPESTRY TRANSITION
CASE CALL "00"
MOVE PARAMETERS FROM MEMORY STACK TO REGISTERS
XD←0
CASE "10" OR "11" RETURN:
MOVE FUNCTION RETURN VALUE FROM X86 HOME TO TAPESTRY HOME
CASE RESUME FROM EXCEPTION
RESTORE TAPESTRY CONTEXT FROM SAVE SLOT
TAPESTRY OS 312

FIG. 3A

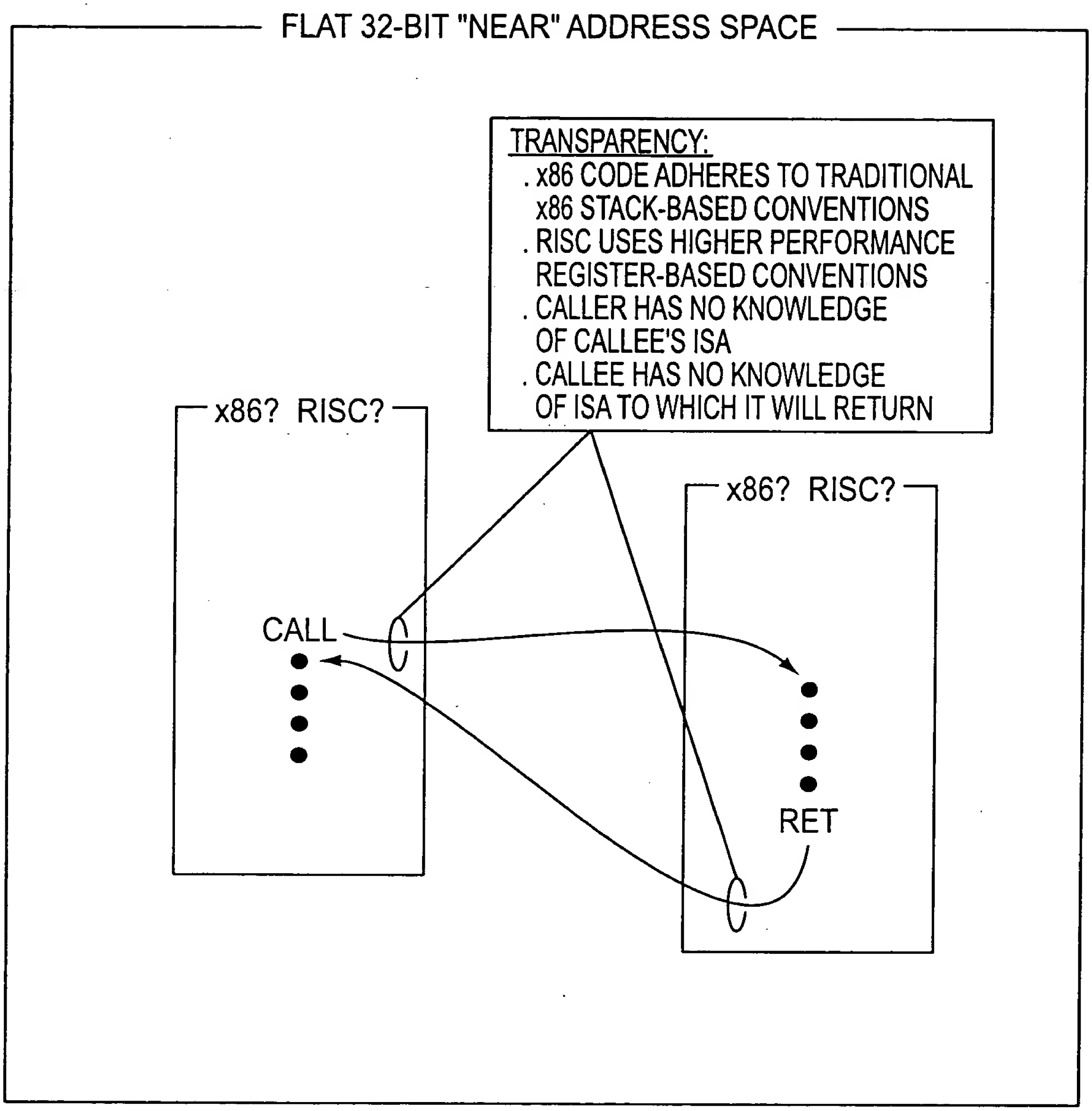

FLAT 32-BIT "NEAR" ADDRESS SPACE
TRANSPARENCY:
. x86 CODE ADHERES TO TRADITIONAL x86 STACK-BASED CONVENTIONS
. RISC USES HIGHER PERFORMANCE REGISTER-BASED CONVENTIONS
. CALLER HAS NO KNOWLEDGE OF CALLEE'S ISA
. CALLEE HAS NO KNOWLEDGE OF ISA TO WHICH IT WILL RETURN
x86? RISC?
CALL
x86? RISC?
RET

FIG. 3B

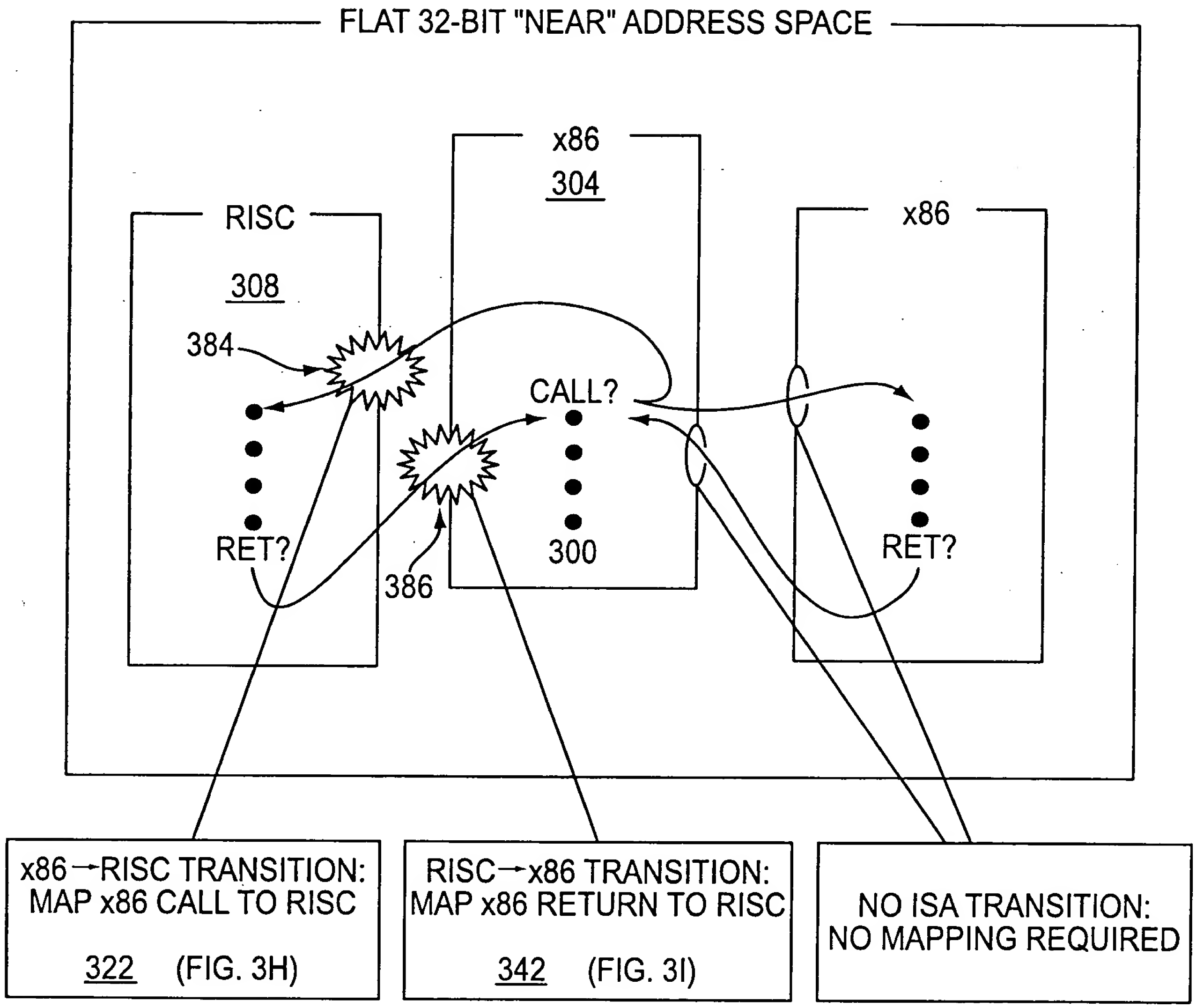

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
304
RISC
308
x86
384
CALL?
RET?
300
386
RET?
x86→RISC TRANSITION: MAP x86 CALL TO RISC
322 (FIG. 3H)
RISC→x86 TRANSITION: MAP x86 RETURN TO RISC
342 (FIG. 3I)
NO ISA TRANSITION: NO MAPPING REQUIRED

FIG. 3C

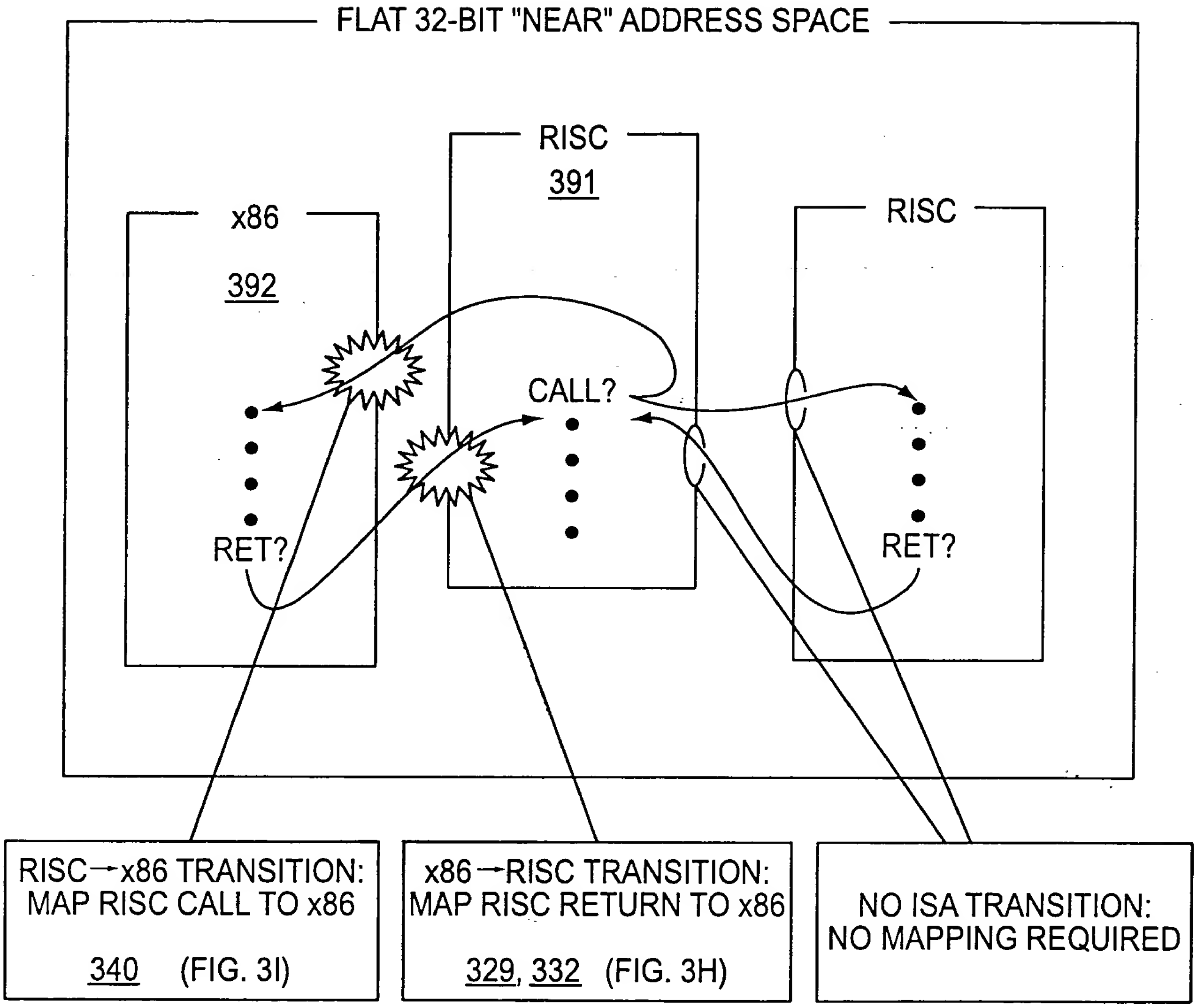

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
392
RISC
391
RISC
CALL?
RET?
RET?
RISC→x86 TRANSITION: MAP RISC CALL TO x86
340 (FIG. 3I)
x86→RISC TRANSITION: MAP RISC RETURN TO x86
329, 332 (FIG. 3H)
NO ISA TRANSITION: NO MAPPING REQUIRED

FIG. 3D

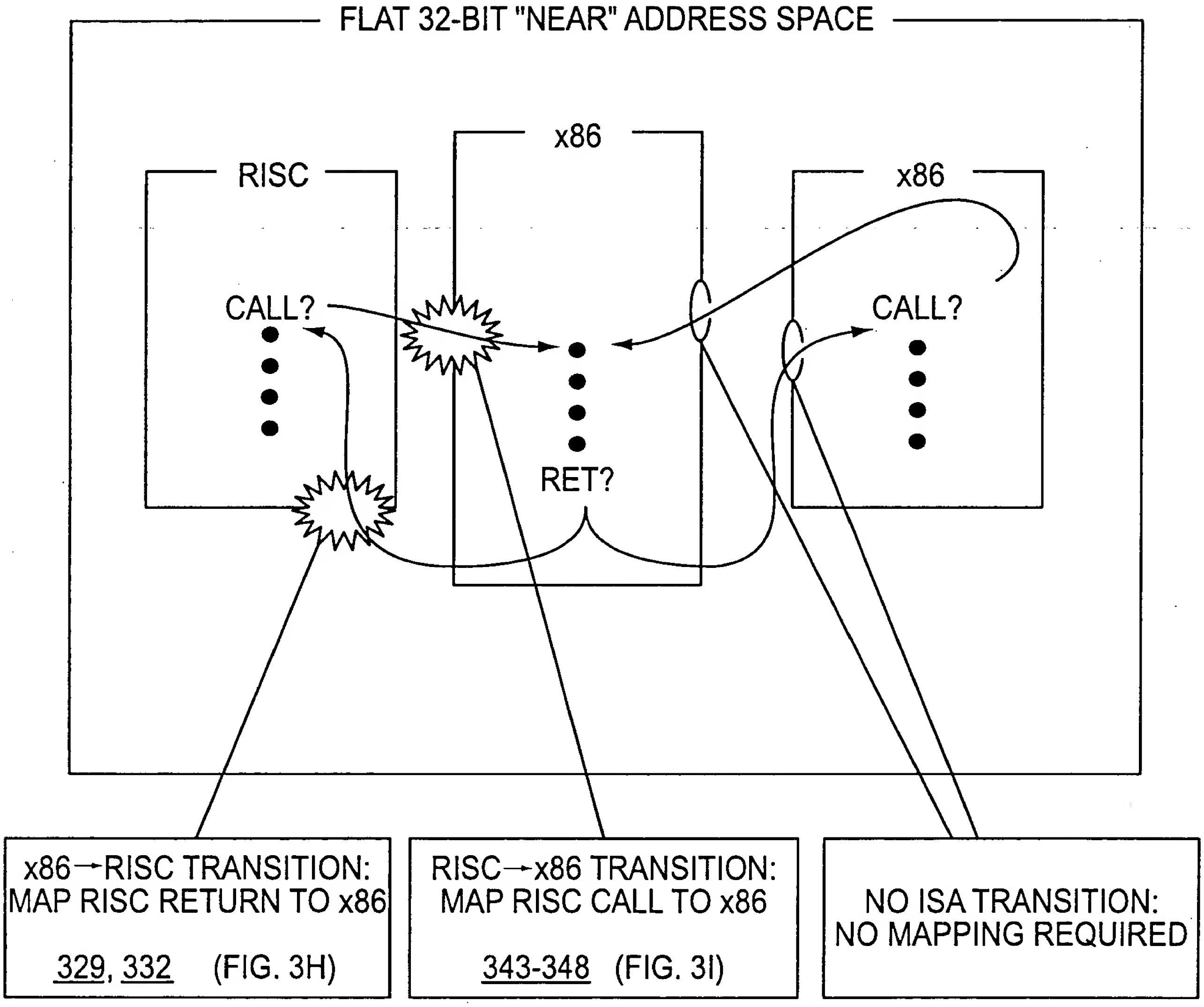

FLAT 32-BIT "NEAR" ADDRESS SPACE
RISC
x86
x86
CALL?
CALL?
RET?
x86→RISC TRANSITION:
MAP RISC RETURN TO x86
329, 332 (FIG. 3H)
RISC→x86 TRANSITION:
MAP RISC CALL TO x86
343-348 (FIG. 3I)
NO ISA TRANSITION:
NO MAPPING REQUIRED

FIG. 3E

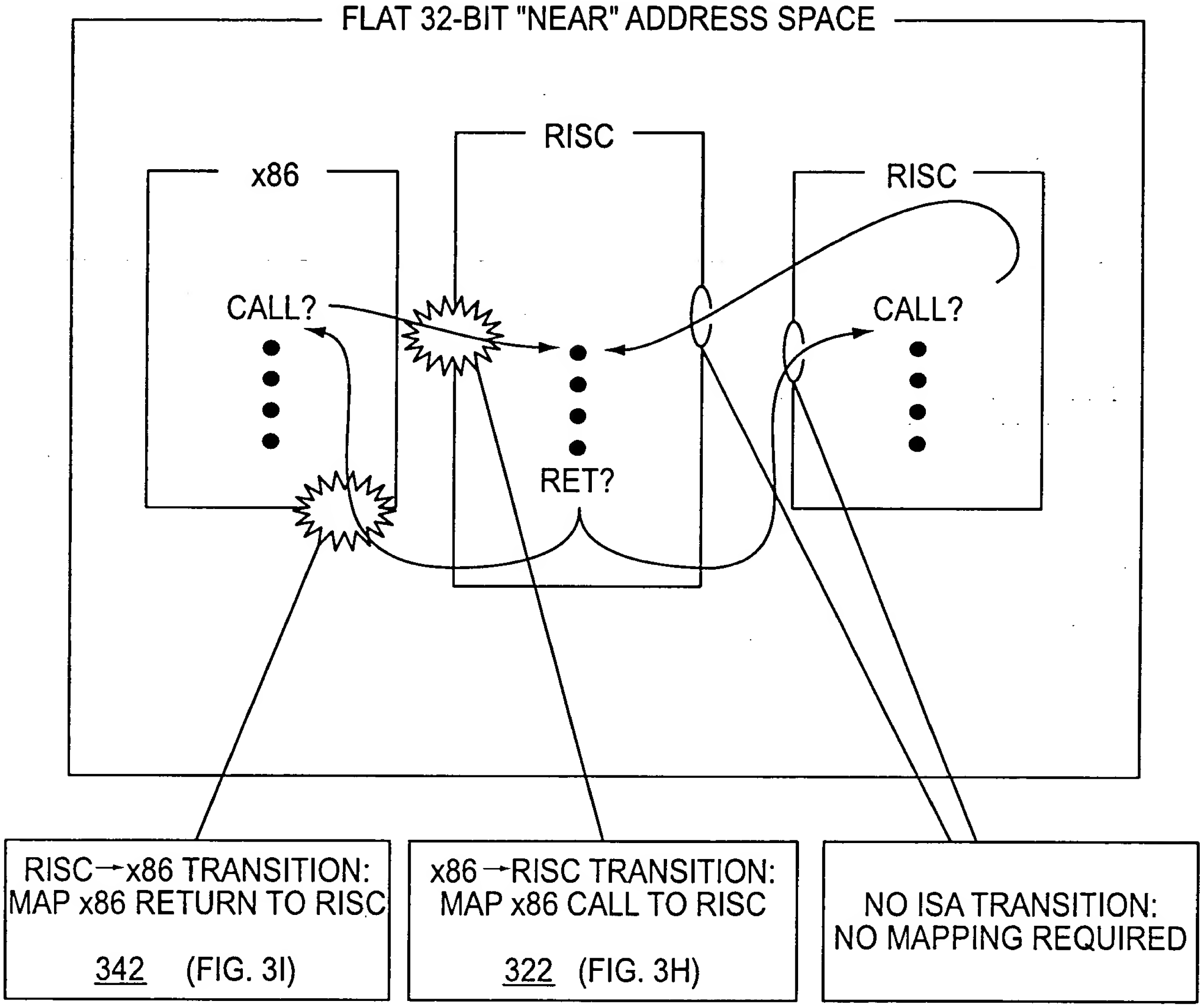

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
RISC
RISC
CALL?
RET?
CALL?
RISC→x86 TRANSITION:
MAP x86 RETURN TO RISC
342 (FIG. 3I)
x86→RISC TRANSITION:
MAP x86 CALL TO RISC
322 (FIG. 3H)
NO ISA TRANSITION:
NO MAPPING REQUIRED

FIG. 3F

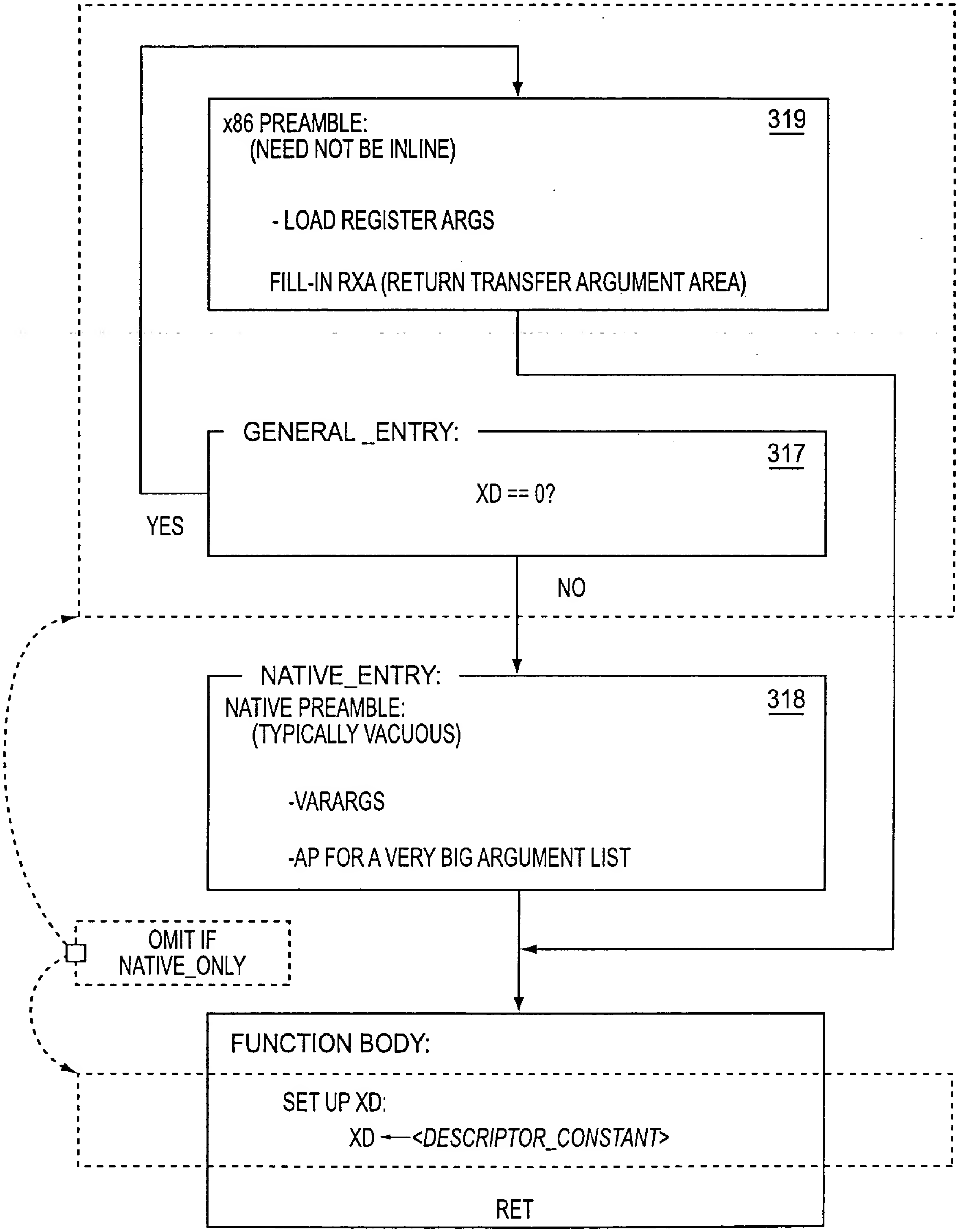

x86 PREAMBLE:
(NEED NOT BE INLINE)
319
- LOAD REGISTER ARGS
FILL-IN RXA (RETURN TRANSFER ARGUMENT AREA)
GENERAL _ENTRY:
317
XD == 0?
YES
NO
NATIVE_ENTRY:
NATIVE PREAMBLE:
(TYPICALLY VACUOUS)
318
-VARARGS
-AP FOR A VERY BIG ARGUMENT LIST
OMIT IF
NATIVE_ONLY
FUNCTION BODY:
SET UP XD:
XD ←<DESCRIPTOR_CONSTANT>
RET

FIG. 3G

320

X86-to Tapestry transition exception handler

```
    // This handler is entered under the following conditions:
    // 1. An x86 caller invokes a native function
    // 2. An x86 function returns to a native caller
    // 3. x86 software returns to or resumes an interrupted native function following
    //    an external asynchronous interrupt, a processor exception, or a context switch
        321
    dispatch on the two least-significant bits of the destination address      {
    case"00"        // calling a native subprogram                                      } 322
        // copy linkage and stack frame information and call parameters from the memory
        // stack to the analogous Tapestry registers
        LR ← [SP++]          // set up linkage register  323
        AP ← SP              // address of first argument  324
        SP ← SP - 8          // allocate return transfer argument area  326
        SP ← SP & (-32)      // round the stack pointer down to a 0 mod 32 boundary  327
        XD ← 0               // inform callee that caller uses X86 calling conventions  328
    case "01"          // resuming an X86 thread suspended during execution of a native routine   } 370
        if the redundant copies of the save slot number in EAX and EDX do not match or if          } 371
              the redundant copies of the timestamp in EBX:ECX and ESI:EDI do not match {
              // some form of bug or thread corruption has been detected
              goto TAPESTRY_CRASH_SYSTEM( thread-corruption-error-code )  372
        }
        save the EBX:ECX timestamp in a 64-bit exception handler temporary register    } 373
               (this will not be overwritten during restoration of the full native context)
        use save slot number in EAX to locate actual save slot storage  374
        restore full entire native context (includes new values for all x86 registers)  375
        if save slot's timestamp does not match the saved timestamp {  376
              // save slot has been reallocated; save slot exhaustion has been detected
              goto TAPESTRY_CRASH_SYSTEM( save-slot-overwritten-error-code )  377
        }
        free the save slot  378
    case"10"           // returning from X86 callee to native caller, result already in registers   } 332
        RV0<63:32> ← edx<31:00>                   // in case result is 64 bits  333
        convert the FP top-of-stack value from 80 bit X86 form to 64-bit form in RVDP  334
        SP ← ESI                                  // restore SP from time of call  337
    case"11"          // returning from X86 callee to native caller, load large result from memory  } 329
        RV0..RV3 ← load 32 bytes from [ESI-32] // (guaranteed naturally aligned)  330
        SP ← ESI                                  // restore SP from time of call  337
    }
    EPC ← EPC & -4          // reset the two low-order bits to zero  336
    RFE  338
```

FIG. 3H

```
                                                                    340
Tapestry-to-X86 transition exception handler
    // This handler is entered under the following conditions:
    // 1. a native caller invokes an x86 function
    // 2. a native function returns to an x86 caller
    switch on XD<3:0> {  341

    XD_RET_FP:                    // result type is floating point
        FO/FI ← FINFLATE.de( RVDP)   // X86 FP results are 80 bits
        SP ← from RXA save           // discard RXA, pad, args
        FPCW ← image after FINIT & push // FP stack has 1 entry
        goto EXIT

    XD_RET_WRITEBACK:              // store result to @RVA, leave RVA in eax
        RVA ← from RXA save          // address of result area
        copy decode(XD<8:4>) bytes from RV0..RV3 to [RVA]                 342
        eax ← RVA                    // X86 expects RVA in eax
        SP ← from RXA save           // discard RXA, pad, args
        FPCW ← image after FINIT          // FP stack is empty
        goto EXIT

    XD_RET_SCALAR:                // result in eax:eda
        edx<31:00> ← eax<63:32>      // in case result is 64 bits
        SP ← from RXA save           // discard RXA, pad, args
        FPCW ← image after FINIT           // FP stack is empty
        goto EXIT

    XD_CALL_HIDDEN_TEMP:  // allocate 32 byte aligned hidden temp   343
        esi ← SP                     // stack cut back on return
        SP ← SP - 32                 // allocate max size temp        } 344
        RVA ← SP                     // RVA consumed later by RR      }
        LR<1:0> ← "11"               // flag address for return & reload   345
        goto CALL_COMMON

    default:                      // remaining XD_CALL_xxx encodings
        esi ← SP                     // stack cut back on return   343
        LR<1:0> ← "10"               // flag address for return   346
CALL_COMMON:                                    347
        interpret XD to push and/or reposition args
        [--SP] ← LR                  // push LR as return address  }
EXIT:                                                              } 348
        setup emulator context and profiling ring buffer pointer   }
    }
    RFE  349                         // to original target
}
```

FIG. 3I

```
                                                                                        350
interrupt/exception handler of Tapestry operating system:
    // Control vectors here when a synchronous exception or asynchronous interrupt is to be
    // exported to / manifested in an x86 machine.

// The interrupt is directed to something within the virtual X86, and thus there is a possibility
// that the X86 operating system will context switch.  So we need to distinguish two cases:
//   either the running process has only X86 state that is relevant to save, or
//   there is extended state that must be saved and associated with the current machine context
//         (e.g., extended state in a Tapestry library call in behalf of a process managed by X86 OS)
if execution was interrupted in the converter – EPC.ISA == X86 {
        // no dependence on extended/native state possible, hence no need to save any      351
        goto EM86_Deliver_Interrupt( interrupt-byte )
} else if EPC.Taxi_Active {
        // A Taxi translated version of some X86 code was running.  Taxi will rollback to an
        // x86 instruction boundary.  Then, if the rollback was induced by an asynchronous external
        // interrupt, Taxi will deliver the appropriate x86 interrupt.  Else, the rollback was induced
        // by a synchronous event so Taxi will resume execution in the converter, retriggering the      353
        // exception but this time with EPC.ISA == X86
        goto TAXi_Rollback( asynchronous-flag, interrupt-byte )
} else if EPC.EM86 {
        // The emulator has been interrupted.  The emulator is coded to allow for such
        // conditions and permits re-entry during long running routines (e.g. far call through a gate)      354
        // to deliver external interrupts
        goto EM86_Deliver_Interrupt( interrupt-byte )
} else {
        // This is the most difficult case - the machine was executing native Tapestry code on
        // behalf of an X86 thread.  The X86 operating system may context switch.  We must save
        // all native state and be able to locate it again when the x86 thread is resumed.
                  361
        allocate a free save slot; if unavailable free the save slot with oldest timestamp and try again
        save the entire native state (both the X86 and the extended state)      362
        save the X86 EIP in the save slot
        overwrite the two low-order bits of EPC with "01" (will become X86 interrupt EIP)      363
        store the 64-bit timestamp in the save slot, in the X86 EBX:ECX register pair (and,      360
                 for further security, store a redundant copy in the X86 ESI:EDI register pair)      364
        store the a number of the allocated save slot in the X86 EAX register (and, again for
                 further security, store a redundant copy in the X86 EDX register)      365
        goto EM86_Deliver_Interrupt( interrupt-byte )
}                                                   369
```

FIG. 3J

```
typedef struct {
     save_slot_t *        newer,          // pointer to next-most-recently-allocated save slot ⎫
     save_slot_t *        older;          // pointer to next-older save slot                    ⎬379c  ⎫
     unsigned int64       epc;            // saved exception PC/IP                              ⎫      ⎪
     unsigned int64       pcw;            // saved exception PCW (program control word)         ⎪      ⎪
     unsigned int64       registers[63];  // save the 63 writeable general registers            ⎬356   ⎬ 355
     ...                                  // other words of Tapestry context                    ⎭      ⎪
     timestamp_t          timestamp;      // timestamp to detect buffer overrun  358                   ⎪
     int                  save_slot_ID;   // ID number of the save slot  357                           ⎪
     boolean              save_slot_is_full;         // full / empty flag  359                         ⎭
} save_slot_t;


save_slot_t *        save_slot_head;         // pointer to the head of the queue  379a
save_slot_t *        save_slot_tail;         // pointer to the tail of the queue  379b


system initialization
     reserve several pages of unpaged memory for save slots
```

FIG. 3K

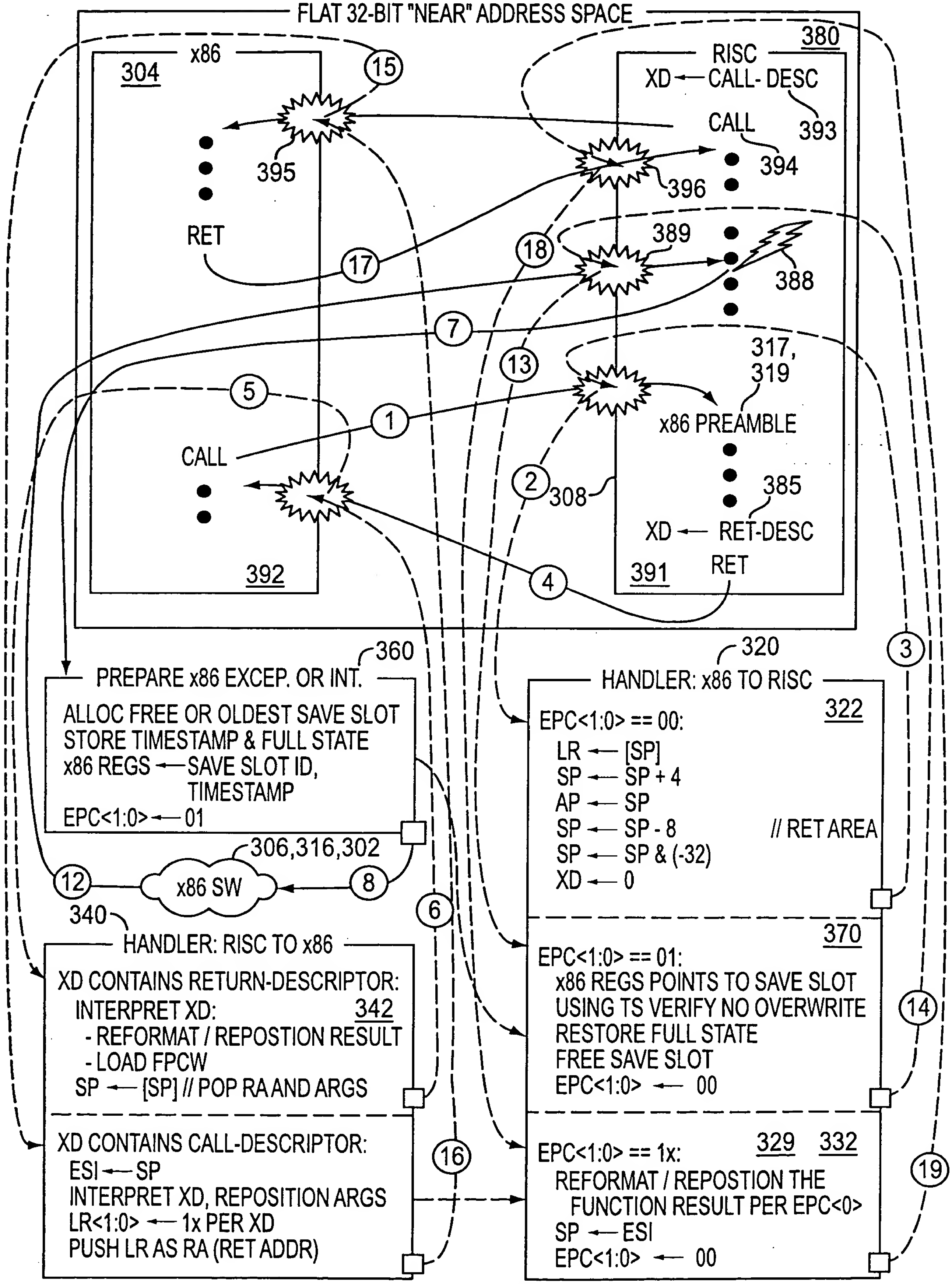

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
304
15
395
RET
CALL
392
RISC
380
XD ← CALL- DESC
393
CALL
394
396
389
388
317,
319
x86 PREAMBLE
385
XD ← RET-DESC
391
RET
308
1
2
4
5
7
13
17
18
360
PREPARE x86 EXCEP. OR INT.
ALLOC FREE OR OLDEST SAVE SLOT
STORE TIMESTAMP & FULL STATE
x86 REGS ← SAVE SLOT ID,
TIMESTAMP
EPC<1:0> ← 01
306,316,302
12
x86 SW
8
6
340
HANDLER: RISC TO x86
XD CONTAINS RETURN-DESCRIPTOR:
INTERPRET XD:
342
- REFORMAT / REPOSTION RESULT
- LOAD FPCW
SP ← [SP] // POP RA AND ARGS
XD CONTAINS CALL-DESCRIPTOR:
ESI ← SP
INTERPRET XD, REPOSITION ARGS
LR<1:0> ← 1x PER XD
PUSH LR AS RA (RET ADDR)
16
320
HANDLER: x86 TO RISC
322
EPC<1:0> == 00:
LR ← [SP]
SP ← SP + 4
AP ← SP
SP ← SP - 8
// RET AREA
SP ← SP & (-32)
XD ← 0
3
370
EPC<1:0> == 01:
x86 REGS POINTS TO SAVE SLOT
USING TS VERIFY NO OVERWRITE
RESTORE FULL STATE
FREE SAVE SLOT
EPC<1:0> ← 00
14
329
332
EPC<1:0> == 1x:
REFORMAT / REPOSTION THE
FUNCTION RESULT PER EPC<0>
SP ← ESI
EPC<1:0> ← 00
19

FIG. 3L

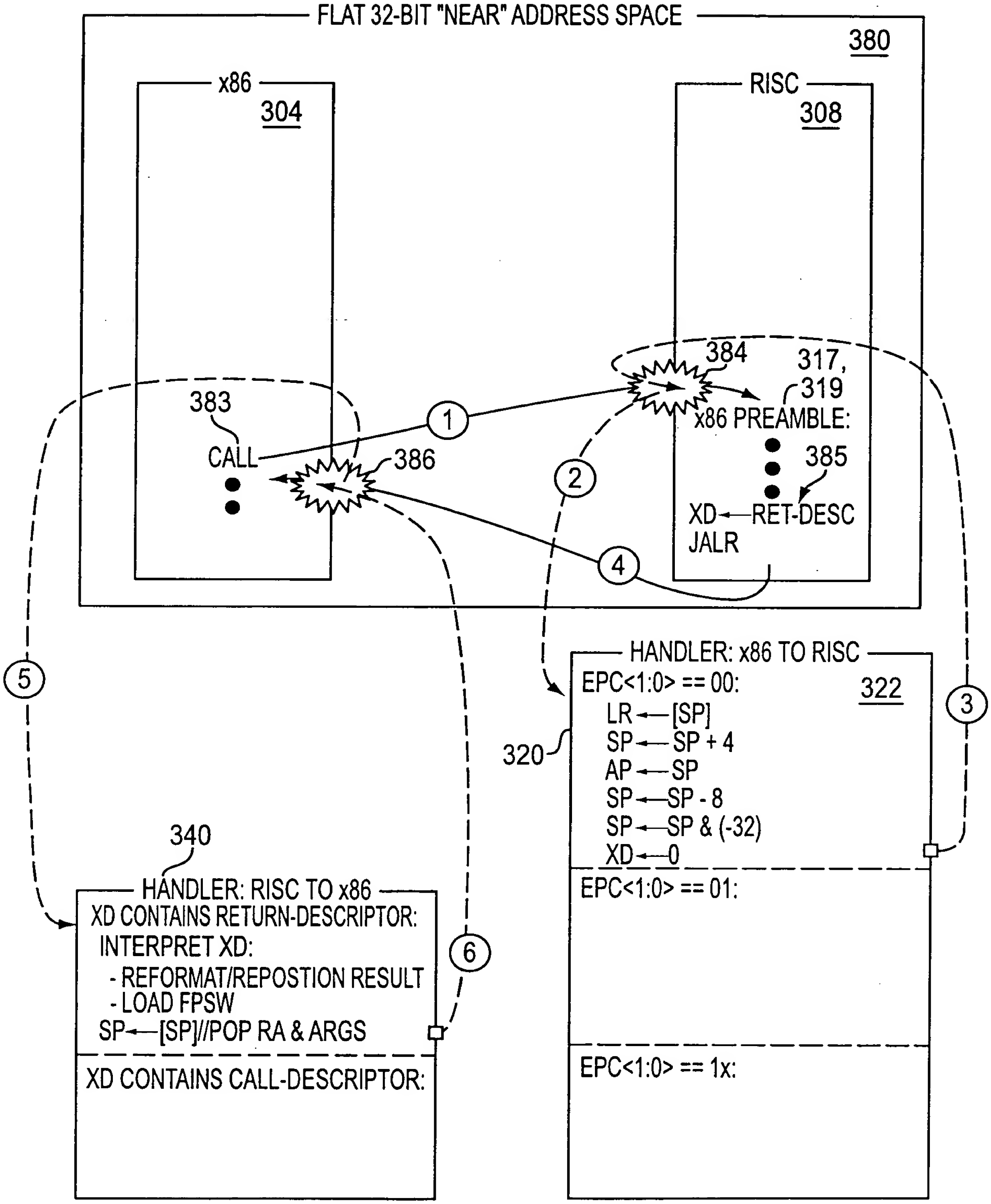

FLAT 32-BIT "NEAR" ADDRESS SPACE
380
x86
304
RISC
308
383
CALL
386
384
317,
319
x86 PREAMBLE:
385
XD←RET-DESC
JALR
1
2
3
4
5
6
HANDLER: x86 TO RISC
EPC<1:0> == 00:
322
320
LR←[SP]
SP←SP + 4
AP←SP
SP←SP - 8
SP←SP & (-32)
XD←0
EPC<1:0> == 01:
EPC<1:0> == 1x:
340
HANDLER: RISC TO x86
XD CONTAINS RETURN-DESCRIPTOR:
INTERPRET XD:
- REFORMAT/REPOSTION RESULT
- LOAD FPSW
SP←[SP]//POP RA & ARGS
XD CONTAINS CALL-DESCRIPTOR:

FIG. 3M

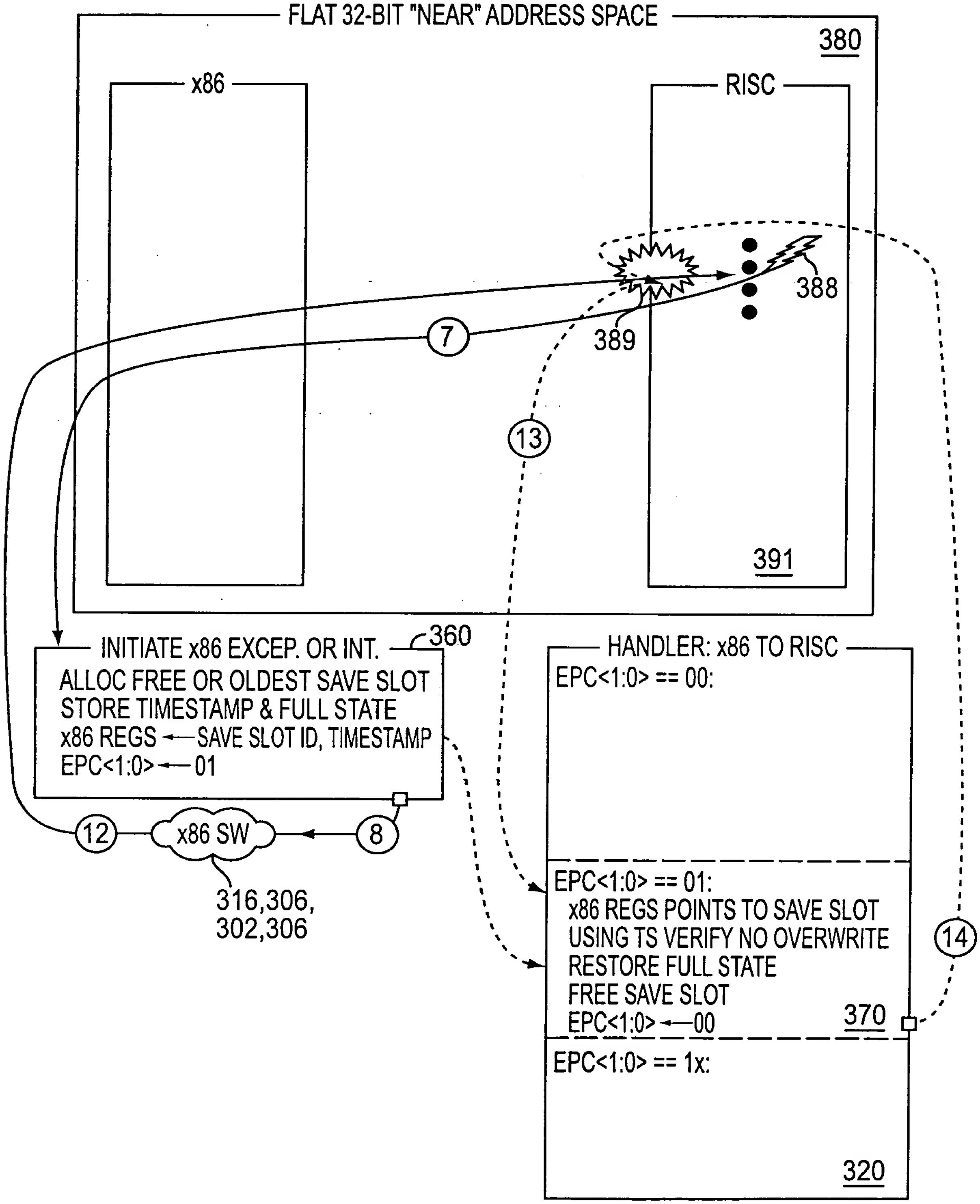

FLAT 32-BIT "NEAR" ADDRESS SPACE
380
x86
RISC
388
389
7
13
391
INITIATE x86 EXCEP. OR INT.
360
ALLOC FREE OR OLDEST SAVE SLOT
STORE TIMESTAMP & FULL STATE
x86 REGS ←— SAVE SLOT ID, TIMESTAMP
EPC<1:0> ←— 01
12
x86 SW
8
316,306,
302,306
HANDLER: x86 TO RISC
EPC<1:0> == 00:
EPC<1:0> == 01:
x86 REGS POINTS TO SAVE SLOT
USING TS VERIFY NO OVERWRITE
RESTORE FULL STATE
FREE SAVE SLOT
EPC<1:0> ←— 00
370
14
EPC<1:0> == 1x:
320

FIG. 3N

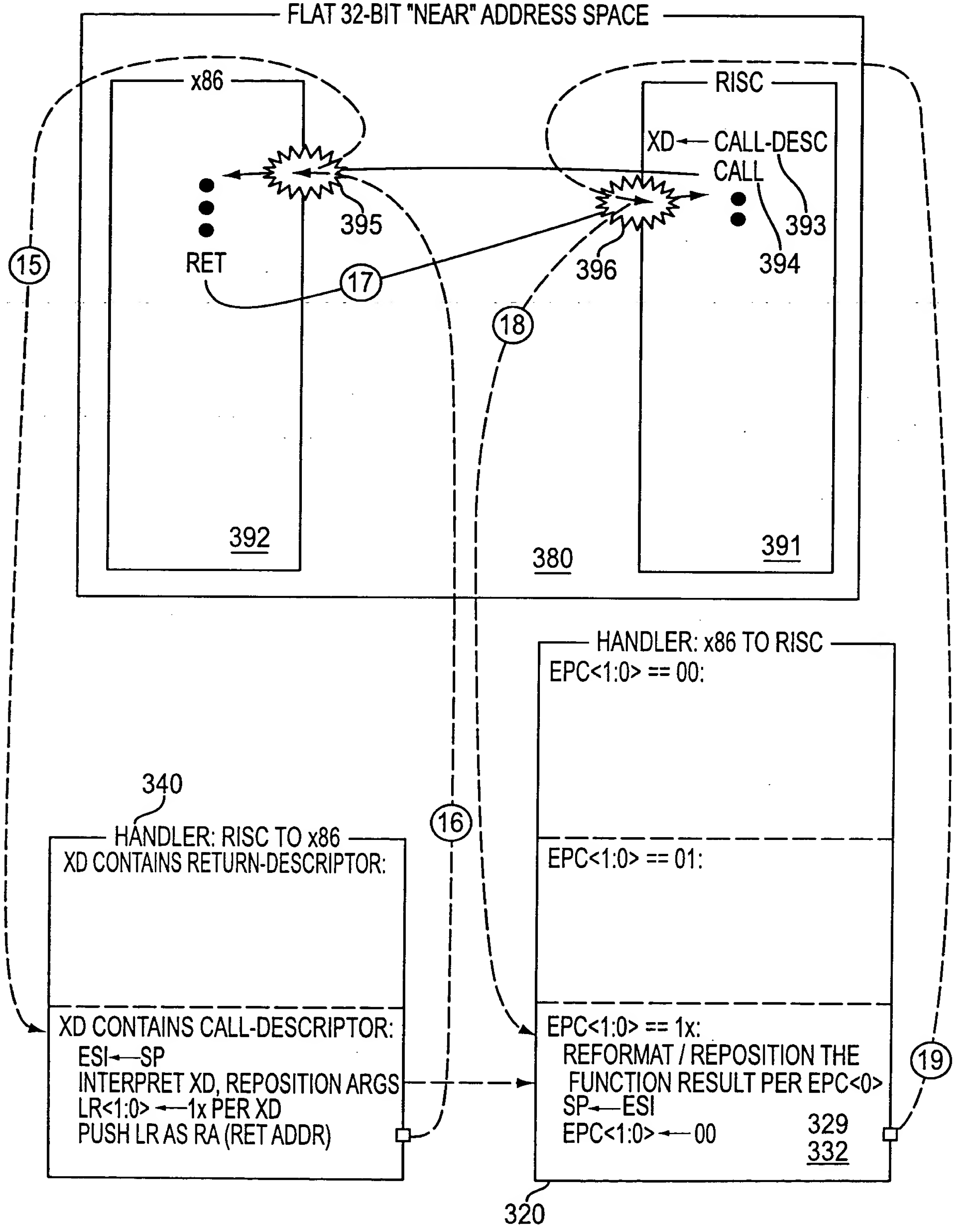
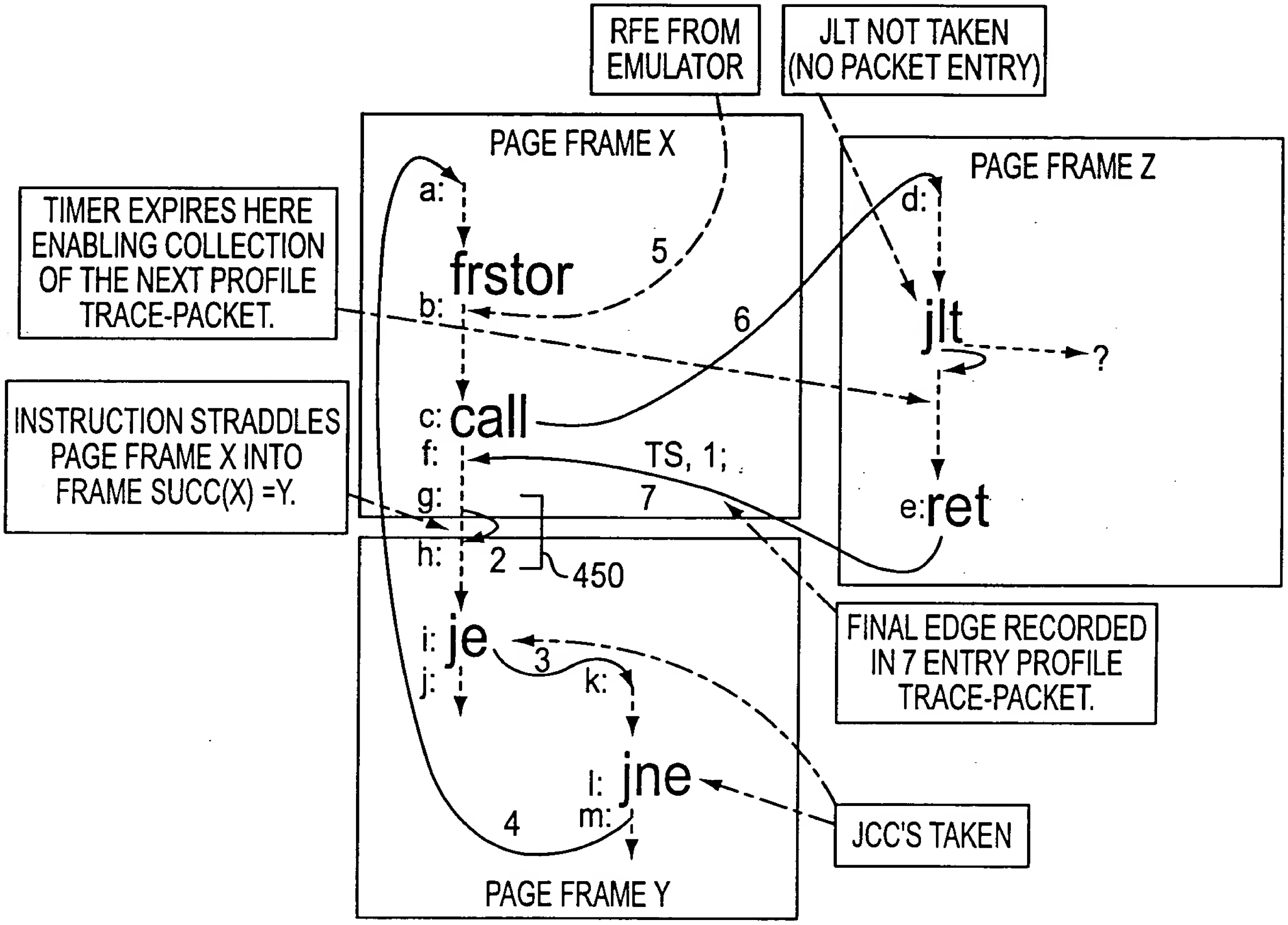

FLAT 32-BIT "NEAR" ADDRESS SPACE
x86
RISC
XD ← CALL-DESC
CALL
393
394
395
396
RET
15
17
18
392
380
391
HANDLER: x86 TO RISC
EPC<1:0> == 00:
340
HANDLER: RISC TO x86
XD CONTAINS RETURN-DESCRIPTOR:
16
EPC<1:0> == 01:
XD CONTAINS CALL-DESCRIPTOR:
ESI ← SP
INTERPRET XD, REPOSITION ARGS
LR<1:0> ← 1x PER XD
PUSH LR AS RA (RET ADDR)
EPC<1:0> == 1x:
REFORMAT / REPOSITION THE
FUNCTION RESULT PER EPC<0>
SP ← ESI
EPC<1:0> ← 00
19
329
332
320

FIG. 3O


RFE FROM EMULATOR
JLT NOT TAKEN (NO PACKET ENTRY)
PAGE FRAME X
PAGE FRAME Z
TIMER EXPIRES HERE ENABLING COLLECTION OF THE NEXT PROFILE TRACE-PACKET.
a:
frstor
5
b:
6
d:
jlt
?
INSTRUCTION STRADDLES PAGE FRAME X INTO FRAME SUCC(X) =Y.
c: call
f:
TS, 1;
g:
7
e:ret
h:
2
450
i: je
j:
3
k:
FINAL EDGE RECORDED IN 7 ENTRY PROFILE TRACE-PACKET.
l: jne
m:
4
JCC'S TAKEN
PAGE FRAME Y

7 ENTRY TRACE PACKET

ENTRY	EVENT CODE	DONE ADDR	NEXT ADDR
	64 BIT TIME STAMP		
1	RET	x86 CONTEXT	*phys X:f*
2	NEW PAGE	*phys Y:g*	*phys Y:h*
3	JCC FORWARD	*phys Y:i*	*phys Y:k*
4	JNZ BACKWARD	*phys Y:l*	*phys X:a*
5	SEQ; ENV CHANGE	x86 CONTEXT	*phys X:b*
6	IP-REL NEAR CALL	*phys X:c*	*phys Z:d*
7	NEAR RET	*phys Z:e*	*phys X:f*

420

430
440, 454
440
440
430
440
440

FIG. 4A

	SOURCE	CODE 402	EVENT	414 REUSE EVENT CODE	PROFILEABLE EVENT 416	INITIATE PACKET 418	PROBEABLE EVENT 610	612 PROBE EVENT BIT-ITLB PROBE ATTRIBUTE OR EMULATOR PROBE
410 412	RFE (CONTEXT_AT_POINT_ENTRY)	0.0000	DEFAULT (x86 TRANSPARENT) EVENT, REUSE ALL CONVERTER VALUES	YES		NO		REUSE EVENT CODE
		0.0001	SIMPLE x86 INSTRUCTION COMPLETION (REUSE EVENT CODE)	YES		NO		REUSE EVENT CODE
		0.0010	PROBE EXCEPTION FAILED	YES		NO		REUSE EVENT CODE
		0.0011	PROBE EXCEPTION FAILED, RELOAD PROBE TIMER	YES		NO		REUSE EVENT CODE
		0.0100	FLUSH EVENT	NO	NO	NO	NO	.
		0.0101	SEQUENTIAL; EXECUTION ENVIRONMENT CHANGED - FORCE EVENT	NO	YES	NO	NO	.
		0.0110	FAR RET	NO	YES	YES	NO	.
		0.0111	IRET	NO	YES	NO	NO	.
		0.1000	FAR CALL	NO	YES	YES	YES	FAR CALL
		0.1001	FAR JMP	NO	YES	YES	NO	.
		0.1010	SPECIAL; EMULATOR EXECUTION, SUPPLY EXTRA INSTRUCTION DATA[a]	NO	YES	NO	NO	.
		0.1011	ABORT PROFILE COLLECTION	NO	NO	NO	NO	.
		0.1100	x86 SYNCHRONOUS/ ASYNCHRONOUS INTERRUPT W/PROBE (GRP 0)	NO	YES	YES	YES	EMULATOR PROBE
		0.1101	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT (GRP 0)	NO	YES	YES	NO	.
		0.1110	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT W/PROBE (GRP 1)	NO	YES	YES	YES	EMULATOR PROBE
		0.1111	x86 SYNCHRONOUS/ASYNCHRONOUS INTERRUPT (GRP 1)	NO	YES	YES	NO	.
404	CONVERTER (NEAR_EDGE_ENTRY)	1.0000	IP-RELATIVE JNZ FORWARD (OPCODE: 75, OF 85)	NO	YES	YES	NO	.
		1.0001	IP-RELATIVE JNZ BACKWARD (OPCODE: 75, OF 85)	NO	YES	YES	YES	JNZ
		1.0010	IP-RELATIVE CONDITIONAL JUMP FORWARD - (JCC, JCXZ, LOOP)	NO	YES	YES	NO	.
		1.0011	IP-RELATIVE CONDITIONAL JUMP BACKWARD - (JCC, JCXZ, LOOP)	NO	YES	YES	YES	COND JUMP
		1.0100	IP-RELATIVE, NEAR JMP FORWARD (OPCODE: E9, EB)	NO	YES	YES	NO	.
		1.0101	IP-RELATIVE, NEAR JMP BACKWARD (OPCODE: E9, EB)	NO	YES	YES	YES	NEAR JUMP
		1.0110	RET/RET IMM16 (OPCODE C3, C2 /W)	NO	YES	YES	NO	.
		1.0111	IP-RELATIVE, NEAR CALL (OPCODE: E8)	NO	YES	YES	YES	NEAR CALL
		1.1000	REPE/REPNE CMPS/SCAS (OPCODE: A6, A7, AE, AF)	NO	YES	NO	NO	.
		1.1001	REP MOVS/STOS/LDOS (OPCODE: A4, A5, AA, AB, AC, AD)	NO	YES	NO	NO	.
		1.1010	INDIRECT NEAR JMP (OPCODE: FF /4)	NO	YES	YES	NO	.
		1.1011	INDIRECT NEAR CALL (OPCODE: FF /2)	NO	YES	YES	YES	NEAR CALL
		1.1100	LOAD FROM I/O MEMORY (TLB.ASI !=0) (NOT USED IN T1)	NO	YES	NO	NO	.
		1.1101	AVAILABLE FOR EXPANSION	NO	NO	NO	NO	.
		1.1110	DEFAULT CONVERTER EVENT; SEQUENTIAL 406	NO	NO	NO	NO	.
		1.1111	NEW PAGE (INSTRUCTION ENDS ON LAST BYTE OF A PAGE FRAME OR STRADDLES ACROSS A PAGE FRAME BOUNDARY) 408	NO	YES	NO	NO	.

FIG. 4B

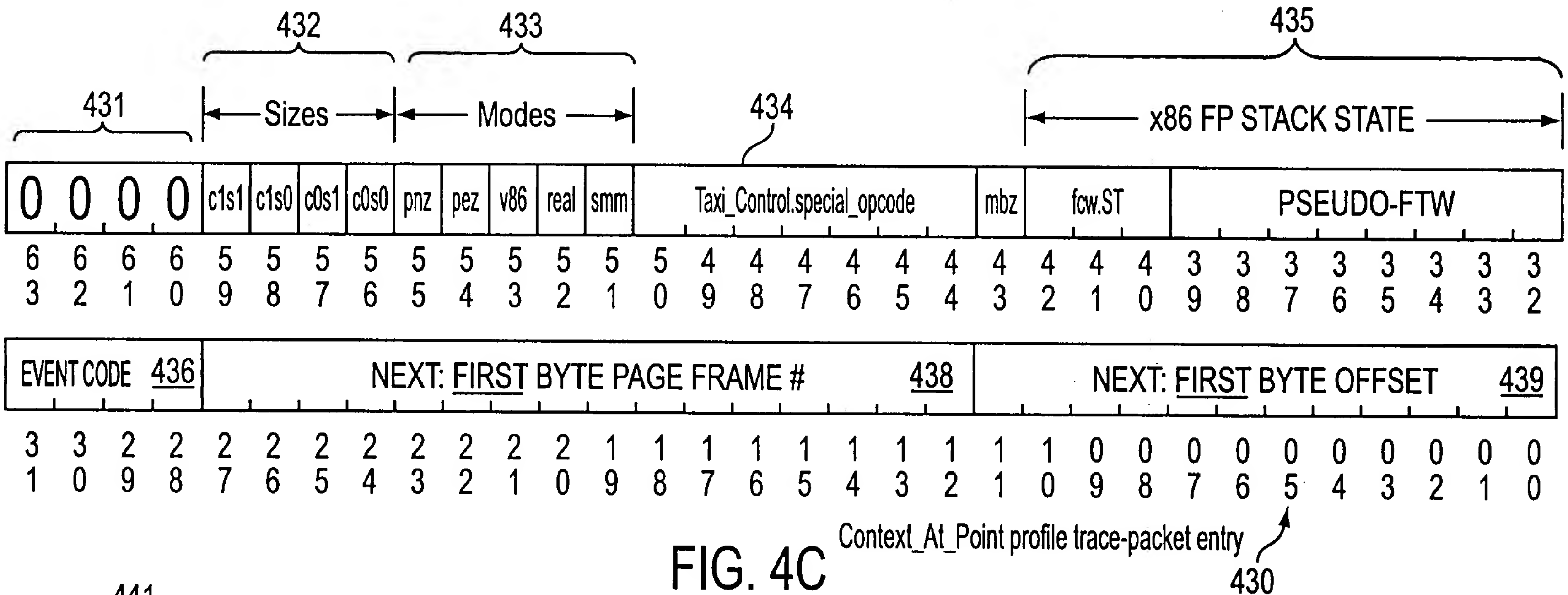

431
432
433
434
435
Sizes
Modes
x86 FP STACK STATE
0 0 0 0
c1s1
c1s0
c0s1
c0s0
pnz
pez
v86
real
smm
Taxi_Control.special_opcode
mbz
fcw.ST
PSEUDO-FTW
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
EVENT CODE 436
NEXT: FIRST BYTE PAGE FRAME # 438
NEXT: FIRST BYTE OFFSET 439
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
Context_At_Point profile trace-packet entry
430

FIG. 4C

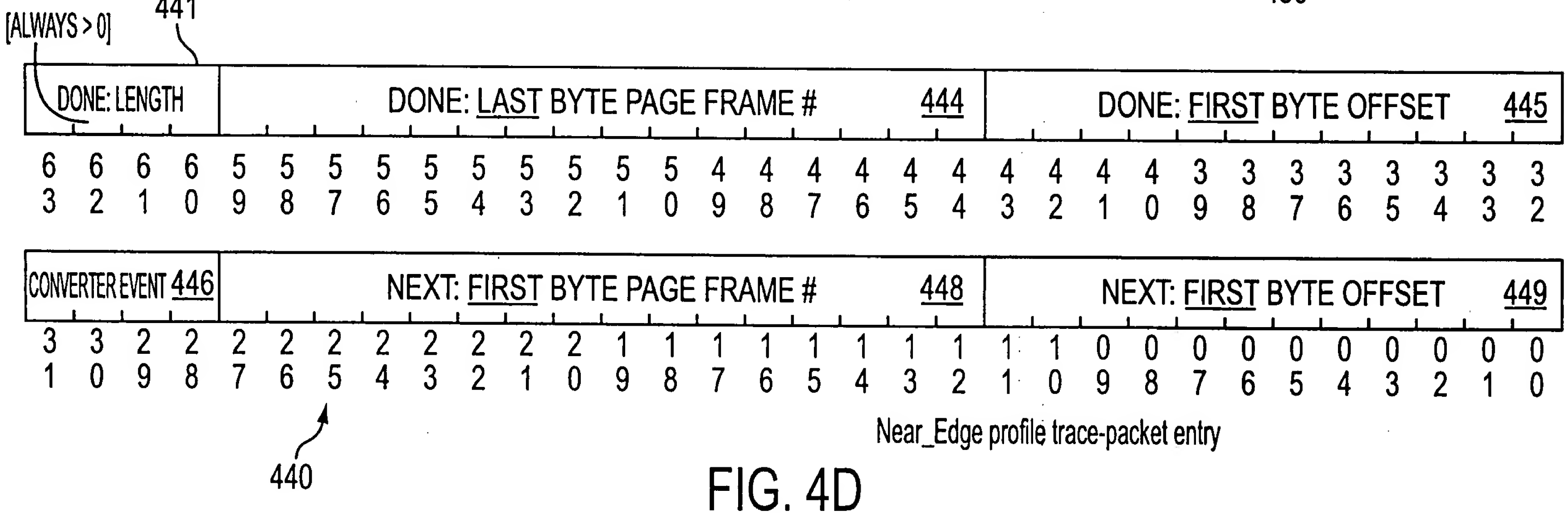

[ALWAYS > 0]
441
DONE: LENGTH
DONE: LAST BYTE PAGE FRAME # 444
DONE: FIRST BYTE OFFSET 445
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
CONVERTER EVENT 446
NEXT: FIRST BYTE PAGE FRAME # 448
NEXT: FIRST BYTE OFFSET 449
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
440
Near_Edge profile trace-packet entry

FIG. 4D

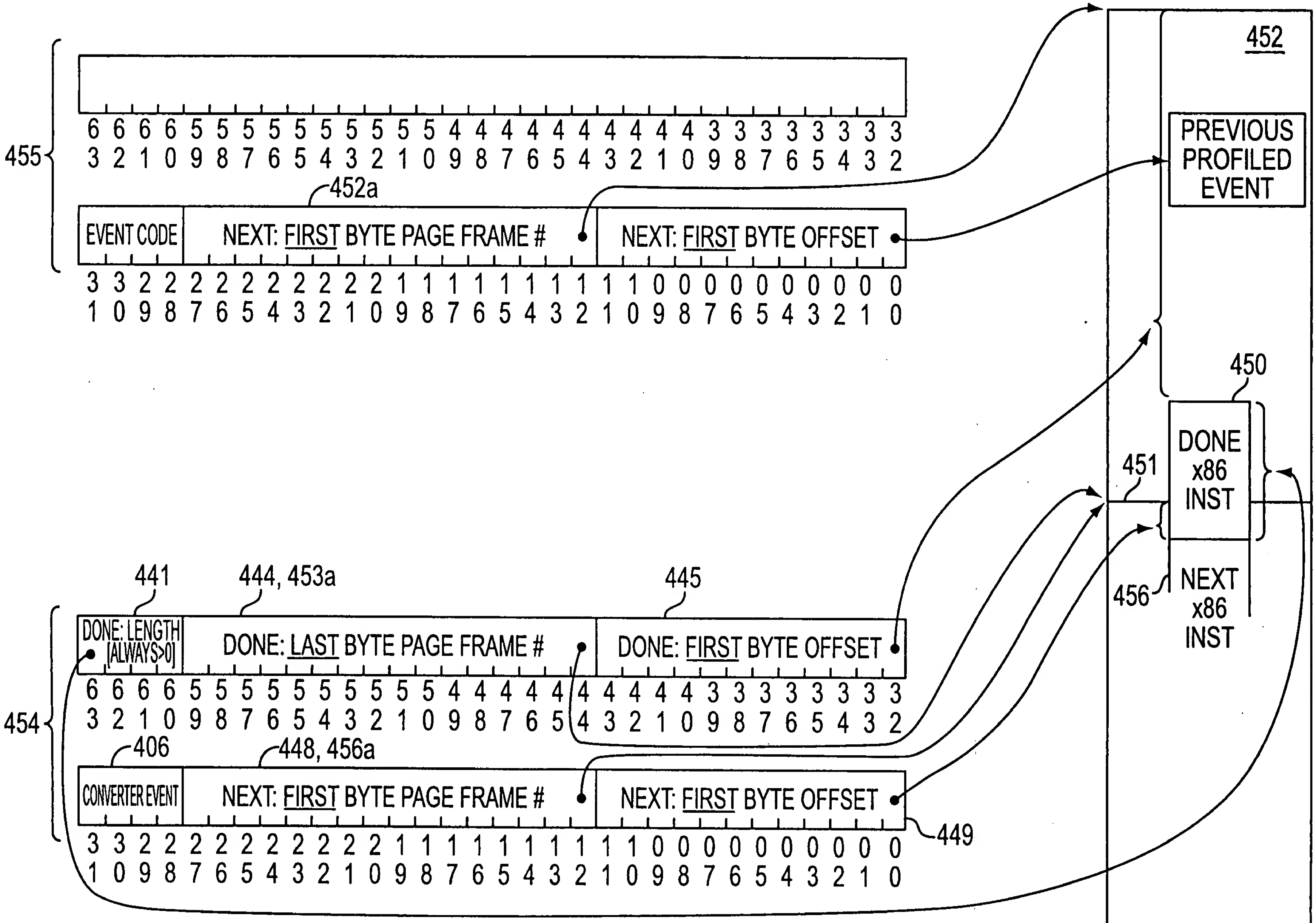

455
EVENT CODE
NEXT: FIRST BYTE PAGE FRAME #
NEXT: FIRST BYTE OFFSET
452a
454
441
DONE: LENGTH [ALWAYS>0]
444, 453a
DONE: LAST BYTE PAGE FRAME #
445
DONE: FIRST BYTE OFFSET
406
CONVERTER EVENT
448, 456a
NEXT: FIRST BYTE PAGE FRAME #
NEXT: FIRST BYTE OFFSET
449
452
PREVIOUS PROFILED EVENT
450
DONE x86 INST
451
456
NEXT x86 INST
453

FIG. 4E

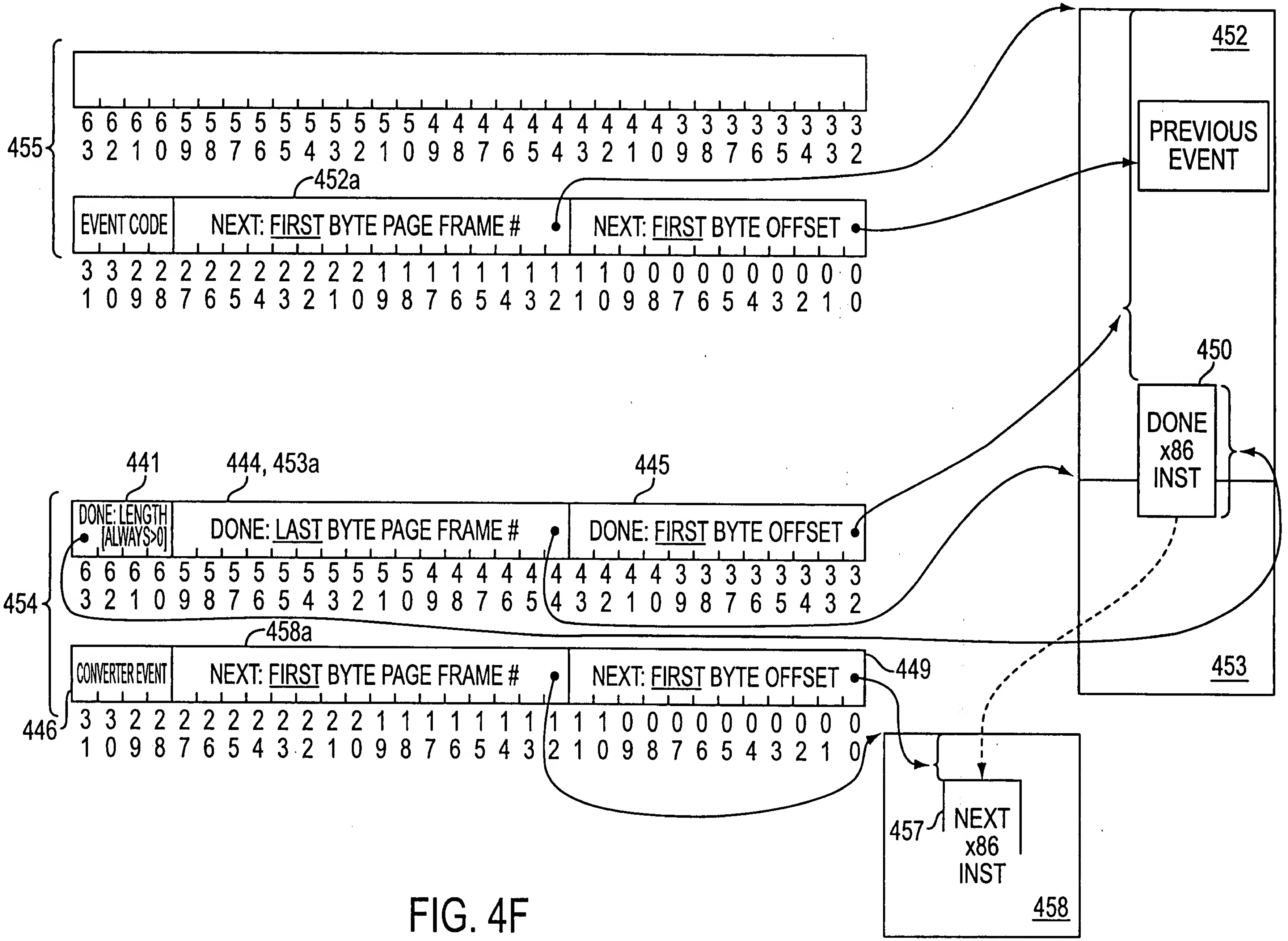
455
6 6 6 6 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2
452a
EVENT CODE
NEXT: FIRST BYTE PAGE FRAME #
NEXT: FIRST BYTE OFFSET
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
452
PREVIOUS EVENT
450
DONE x86 INST
453
441
444, 453a
445
454
DONE: LENGTH [ALWAYS>0]
DONE: LAST BYTE PAGE FRAME #
DONE: FIRST BYTE OFFSET
6 6 6 6 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3
3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2
458a
CONVERTER EVENT
NEXT: FIRST BYTE PAGE FRAME #
NEXT: FIRST BYTE OFFSET
449
446
3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
457
NEXT x86 INST
458

FIG. 4F

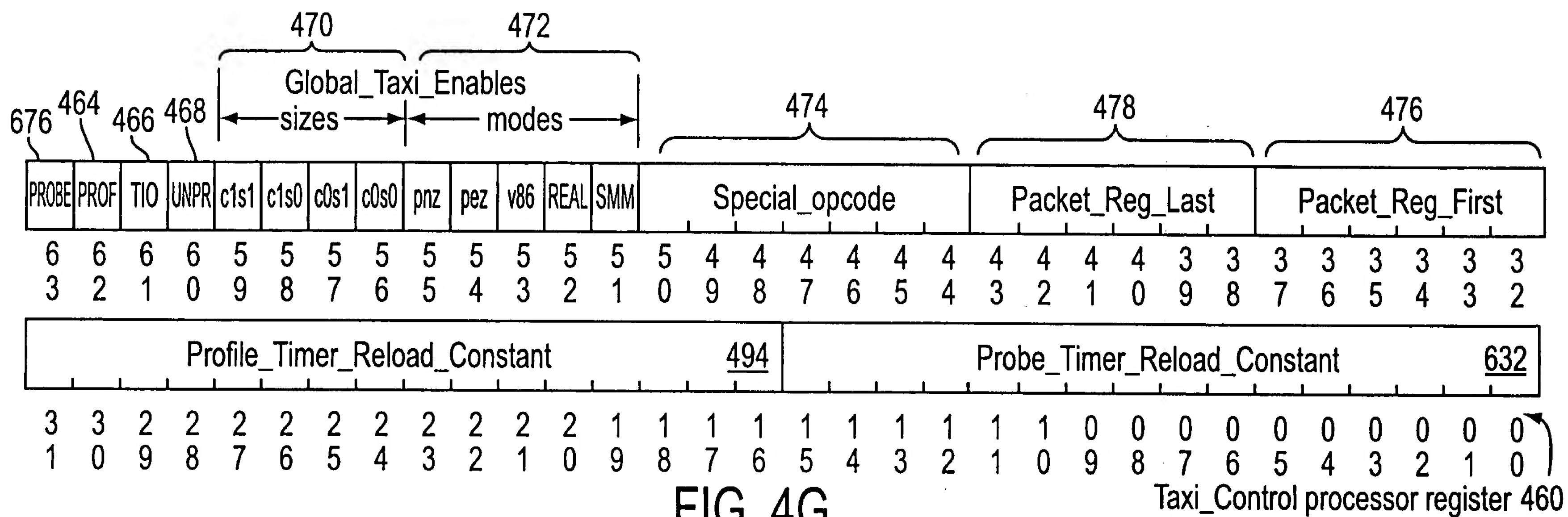


FIG. 4G

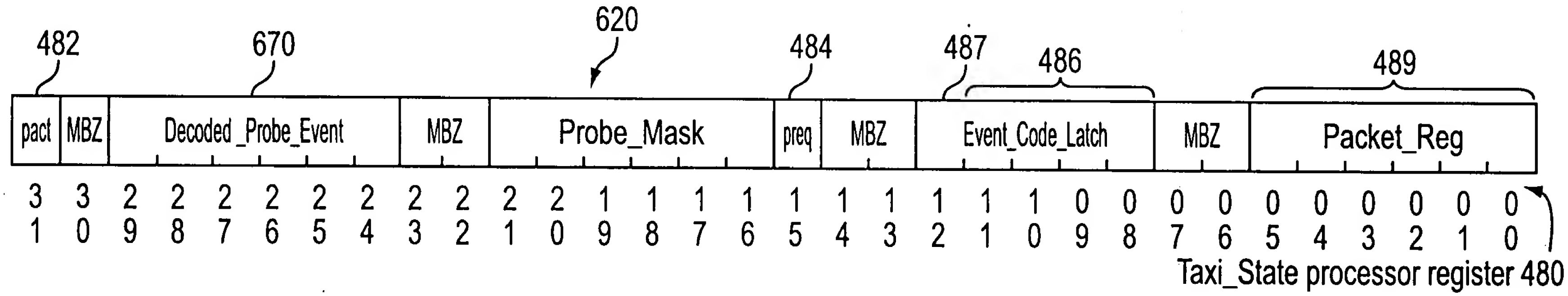


FIG. 4H

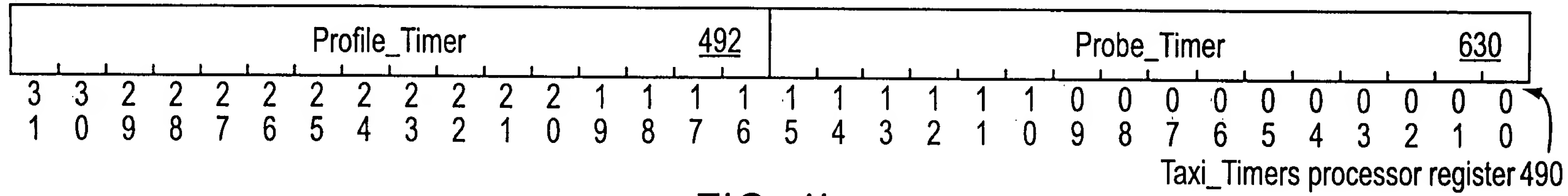


FIG. 4I

EVENTS
pe_{init} = "initiate packet" profile event 418
$pe_{\overline{init}}$ = non-"initiate packet" profile event
pe = any profile event 416
te = timer expiry
ap = abort packet

STATE VARIABLES
PR = Profile_Request flag 488
PA = Profile_Active flag 482

RULES
te EVENT:
 PR ← 1

PR & $\overline{PA}$ & pe_{init}:
 PR ← 0
 PA ← 1
 Init Packet_Reg
 Save Timestamp
 Log Event (CAP)
 Full Packet? / Packet_Reg++

PA & pe:
 Log Event (CAP or NE)
 Full Packet? / Packet_Reg++

FULL PACKET:
 PA ← 0
 profile exception

ap EVENT:
 PA ← 0

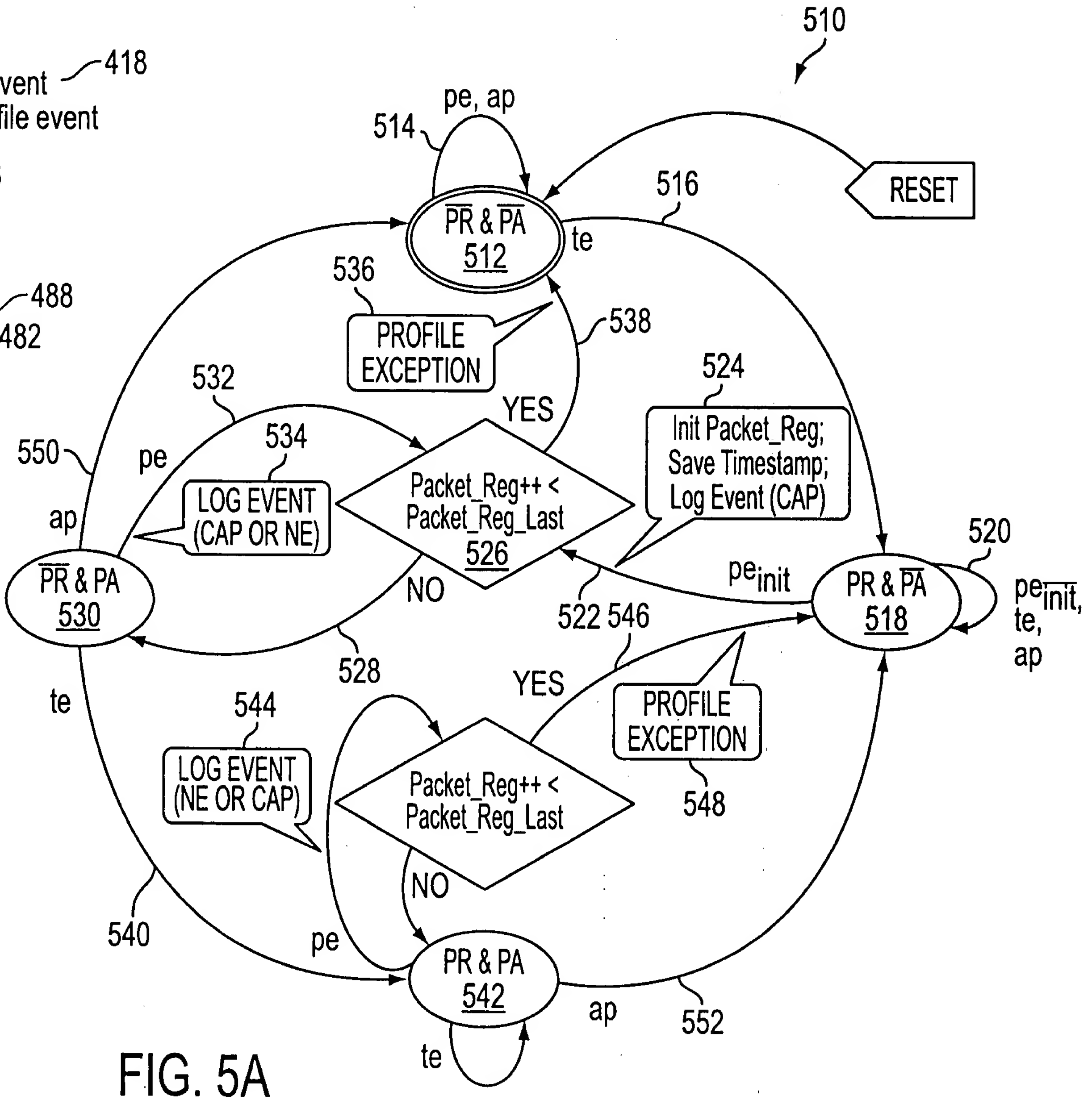


FIG. 5A

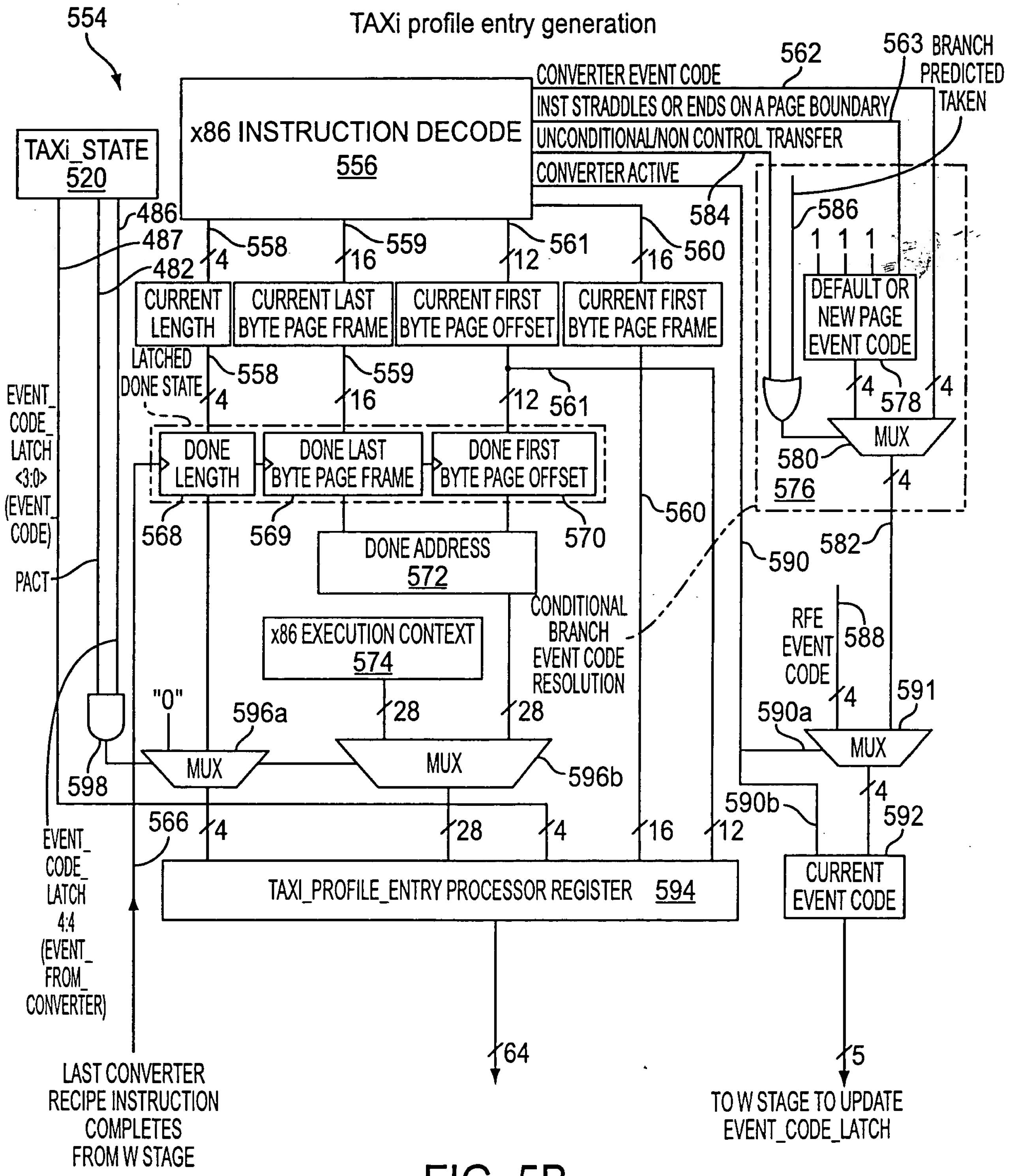

TAXi profile entry generation
554
562
563 BRANCH PREDICTED TAKEN
x86 INSTRUCTION DECODE 556
CONVERTER EVENT CODE
INST STRADDLES OR ENDS ON A PAGE BOUNDARY
UNCONDITIONAL/NON CONTROL TRANSFER
CONVERTER ACTIVE
TAXi_STATE 520
486
487
482
584
586
1 1 1
558
4
559
16
561
12
16
560
CURRENT LENGTH
CURRENT LAST BYTE PAGE FRAME
CURRENT FIRST BYTE PAGE OFFSET
CURRENT FIRST BYTE PAGE FRAME
DEFAULT OR NEW PAGE EVENT CODE
LATCHED DONE STATE
558
4
559
16
12
561
4
578
4
EVENT_CODE_LATCH <3:0> (EVENT_CODE)
DONE LENGTH
DONE LAST BYTE PAGE FRAME
DONE FIRST BYTE PAGE OFFSET
MUX
580
576
4
568
569
570
560
DONE ADDRESS 572
582
590
PACT
CONDITIONAL BRANCH EVENT CODE RESOLUTION
x86 EXECUTION CONTEXT 574
RFE EVENT CODE
588
4
"0"
596a
28
28
590a
591
MUX
MUX
MUX
598
596b
4
590b
592
EVENT_CODE_LATCH 4:4 (EVENT_FROM_CONVERTER)
566
4
28
4
16
12
TAXI_PROFILE_ENTRY PROCESSOR REGISTER 594
CURRENT EVENT CODE
64
5
LAST CONVERTER RECIPE INSTRUCTION COMPLETES FROM W STAGE
TO W STAGE TO UPDATE EVENT_CODE_LATCH

FIG. 5B

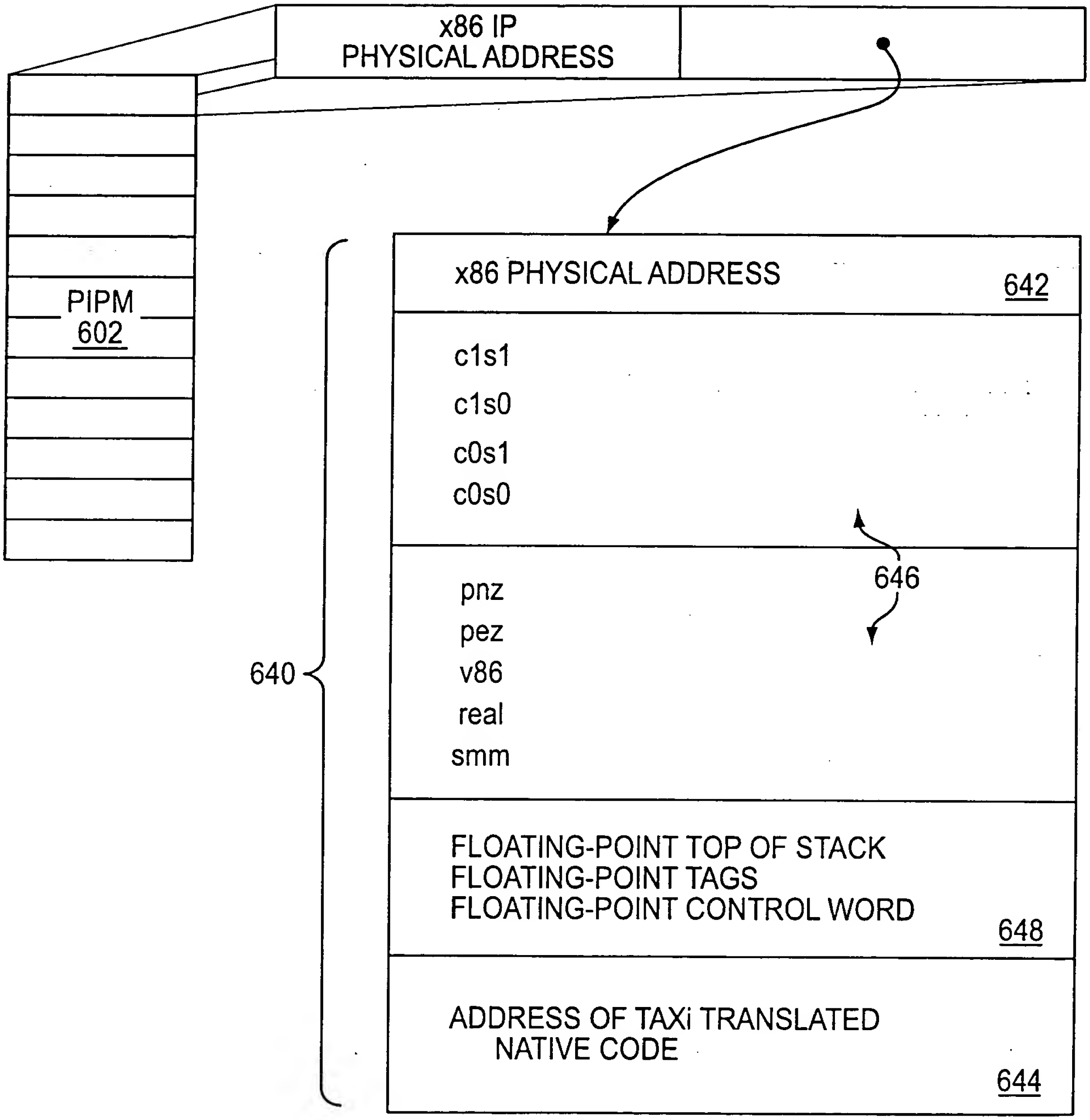

x86 IP
PHYSICAL ADDRESS
PIPM
602
x86 PHYSICAL ADDRESS
642
c1s1
c1s0
c0s1
c0s0
pnz
pez
v86
real
smm
646
640
FLOATING-POINT TOP OF STACK
FLOATING-POINT TAGS
FLOATING-POINT CONTROL WORD
648
ADDRESS OF TAXi TRANSLATED
NATIVE CODE
644

FIG. 6A

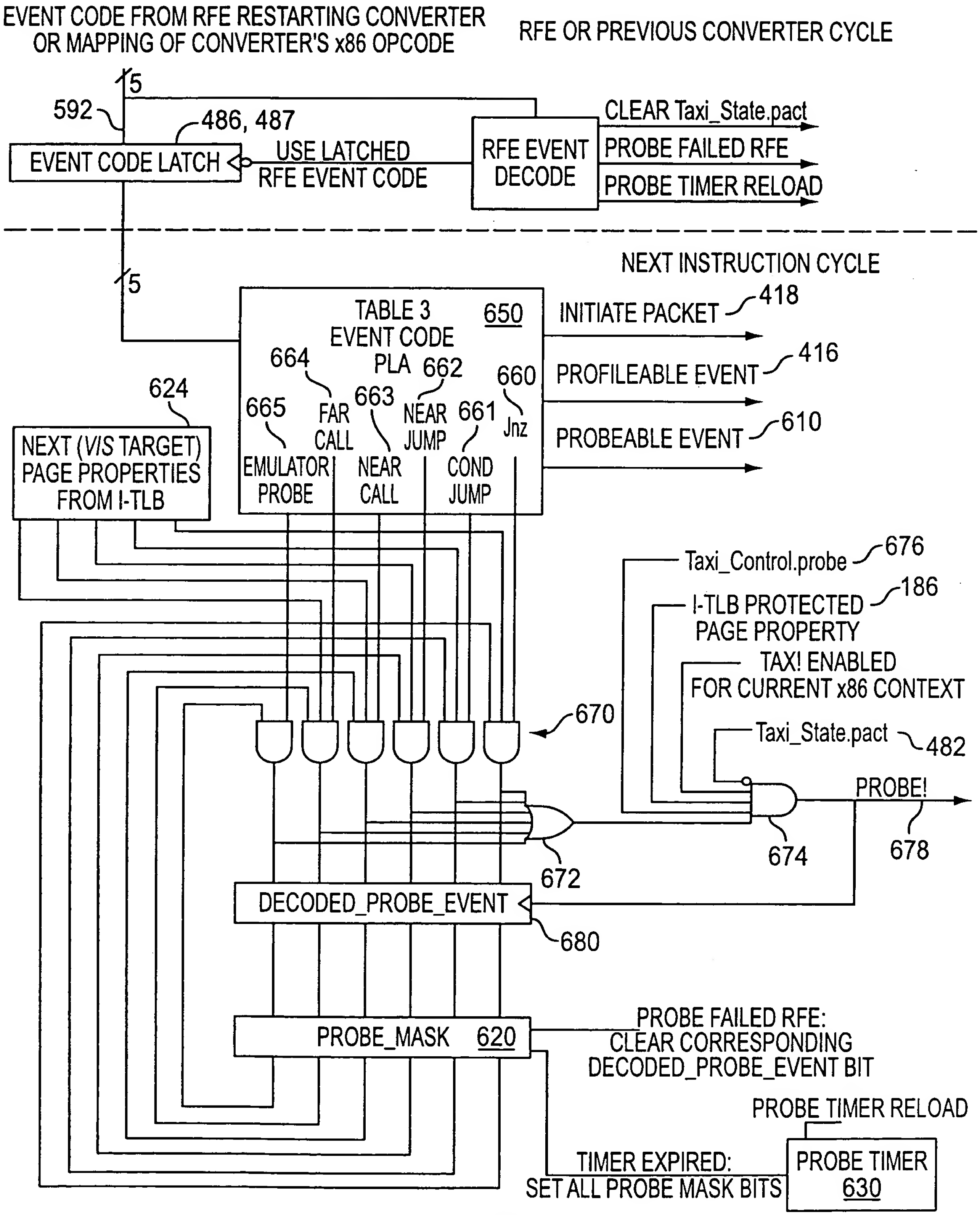

EVENT CODE FROM RFE RESTARTING CONVERTER
OR MAPPING OF CONVERTER'S x86 OPCODE
RFE OR PREVIOUS CONVERTER CYCLE
5
592
486, 487
EVENT CODE LATCH
USE LATCHED
RFE EVENT CODE
RFE EVENT
DECODE
CLEAR Taxi_State.pact
PROBE FAILED RFE
PROBE TIMER RELOAD
NEXT INSTRUCTION CYCLE
5
TABLE 3
EVENT CODE
PLA
650
INITIATE PACKET
418
PROFILEABLE EVENT
416
PROBEABLE EVENT
610
664
662
660
663
661
665
FAR
CALL
NEAR
JUMP
Jnz
EMULATOR
PROBE
NEAR
CALL
COND
JUMP
624
NEXT (V/S TARGET)
PAGE PROPERTIES
FROM I-TLB
Taxi_Control.probe
676
I-TLB PROTECTED
PAGE PROPERTY
186
TAX! ENABLED
FOR CURRENT x86 CONTEXT
Taxi_State.pact
482
670
PROBE!
674
678
672
DECODED_PROBE_EVENT
680
PROBE_MASK
620
PROBE FAILED RFE:
CLEAR CORRESPONDING
DECODED_PROBE_EVENT BIT
PROBE TIMER RELOAD
TIMER EXPIRED:
SET ALL PROBE MASK BITS
PROBE TIMER
630

FIG. 6B

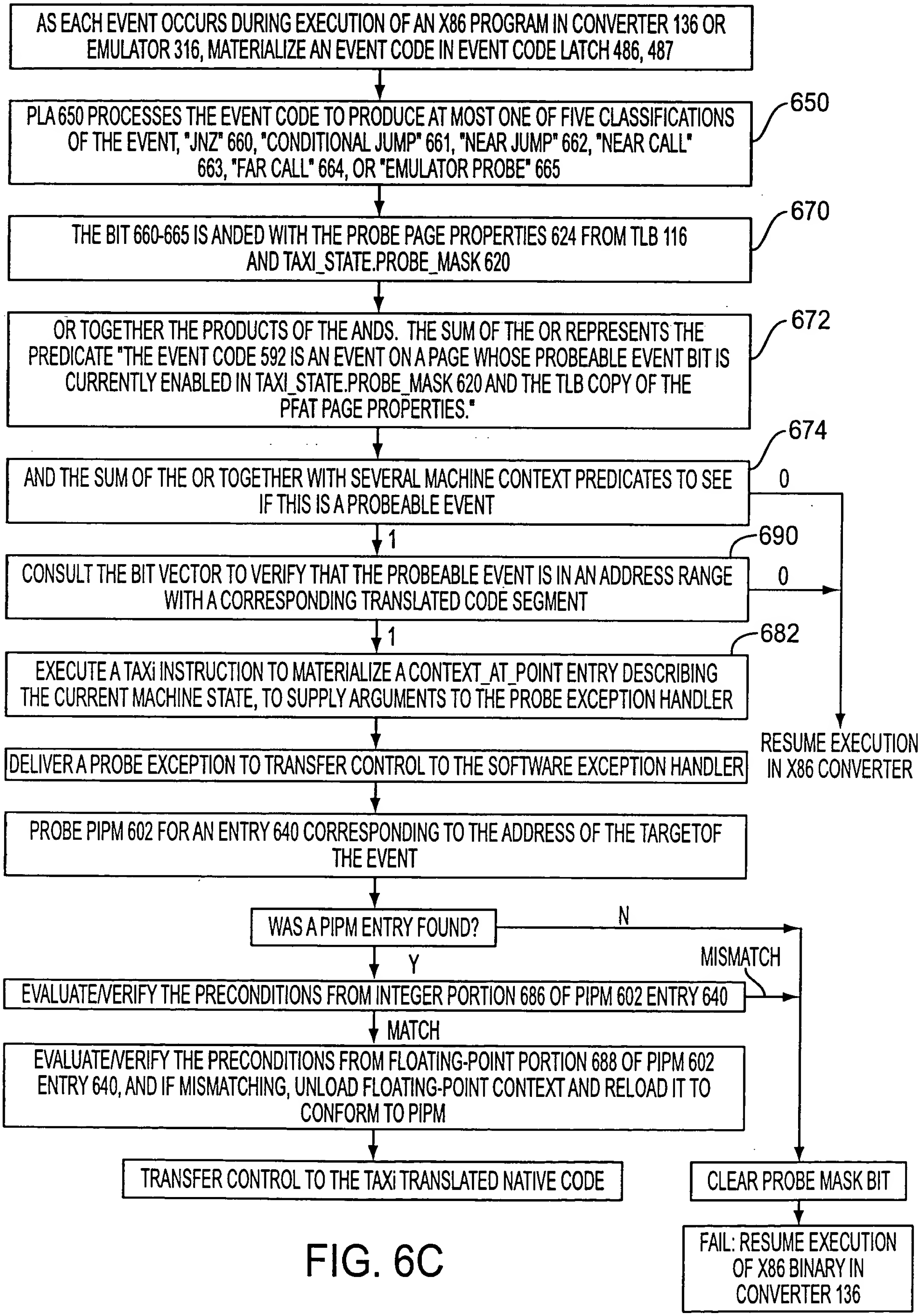
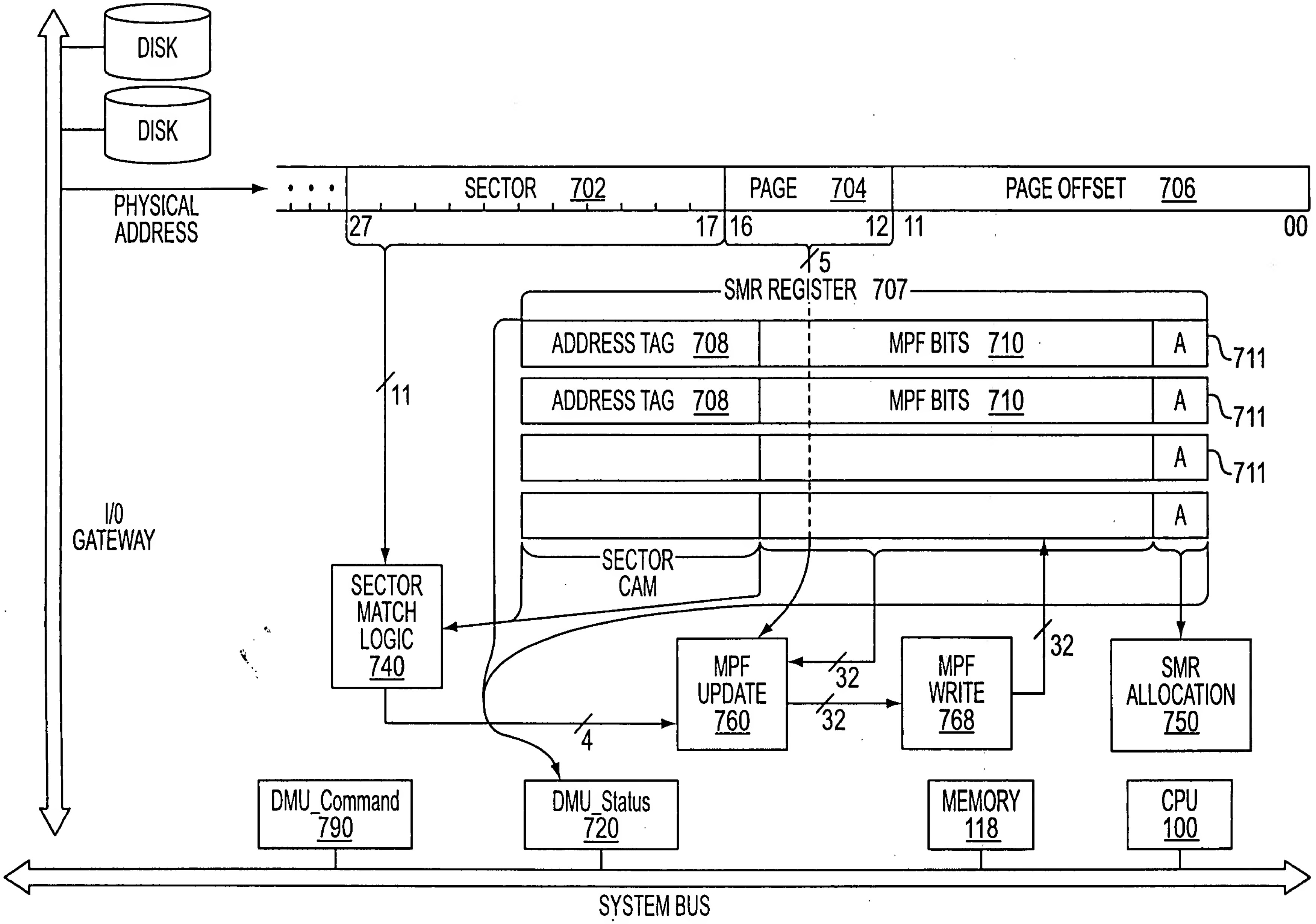

AS EACH EVENT OCCURS DURING EXECUTION OF AN X86 PROGRAM IN CONVERTER 136 OR EMULATOR 316, MATERIALIZE AN EVENT CODE IN EVENT CODE LATCH 486, 487
650
PLA 650 PROCESSES THE EVENT CODE TO PRODUCE AT MOST ONE OF FIVE CLASSIFICATIONS OF THE EVENT, "JNZ" 660, "CONDITIONAL JUMP" 661, "NEAR JUMP" 662, "NEAR CALL" 663, "FAR CALL" 664, OR "EMULATOR PROBE" 665
670
THE BIT 660-665 IS ANDED WITH THE PROBE PAGE PROPERTIES 624 FROM TLB 116 AND TAXI_STATE.PROBE_MASK 620
672
OR TOGETHER THE PRODUCTS OF THE ANDS. THE SUM OF THE OR REPRESENTS THE PREDICATE "THE EVENT CODE 592 IS AN EVENT ON A PAGE WHOSE PROBEABLE EVENT BIT IS CURRENTLY ENABLED IN TAXI_STATE.PROBE_MASK 620 AND THE TLB COPY OF THE PFAT PAGE PROPERTIES."
674
AND THE SUM OF THE OR TOGETHER WITH SEVERAL MACHINE CONTEXT PREDICATES TO SEE IF THIS IS A PROBEABLE EVENT
0
1
690
CONSULT THE BIT VECTOR TO VERIFY THAT THE PROBEABLE EVENT IS IN AN ADDRESS RANGE WITH A CORRESPONDING TRANSLATED CODE SEGMENT
0
1
682
EXECUTE A TAXi INSTRUCTION TO MATERIALIZE A CONTEXT_AT_POINT ENTRY DESCRIBING THE CURRENT MACHINE STATE, TO SUPPLY ARGUMENTS TO THE PROBE EXCEPTION HANDLER
RESUME EXECUTION IN X86 CONVERTER
DELIVER A PROBE EXCEPTION TO TRANSFER CONTROL TO THE SOFTWARE EXCEPTION HANDLER
PROBE PIPM 602 FOR AN ENTRY 640 CORRESPONDING TO THE ADDRESS OF THE TARGETOF THE EVENT
WAS A PIPM ENTRY FOUND?
N
Y
MISMATCH
EVALUATE/VERIFY THE PRECONDITIONS FROM INTEGER PORTION 686 OF PIPM 602 ENTRY 640
MATCH
EVALUATE/VERIFY THE PRECONDITIONS FROM FLOATING-POINT PORTION 688 OF PIPM 602 ENTRY 640, AND IF MISMATCHING, UNLOAD FLOATING-POINT CONTEXT AND RELOAD IT TO CONFORM TO PIPM
TRANSFER CONTROL TO THE TAXi TRANSLATED NATIVE CODE
CLEAR PROBE MASK BIT
FAIL: RESUME EXECUTION OF X86 BINARY IN CONVERTER 136

FIG. 6C



DISK
DISK
PHYSICAL ADDRESS
I/0 GATEWAY
SECTOR 702
PAGE 704
PAGE OFFSET 706
27
17
16
12
11
00
5
11
SMR REGISTER 707
ADDRESS TAG 708
MPF BITS 710
A
711
ADDRESS TAG 708
MPF BITS 710
A
711
A
711
A
SECTOR CAM
SECTOR MATCH LOGIC 740
MPF UPDATE 760
MPF WRITE 768
SMR ALLOCATION 750
4
32
32
32
DMU_Command 790
DMU_Status 720
MEMORY 118
CPU 100
SYSTEM BUS

FIG. 7A

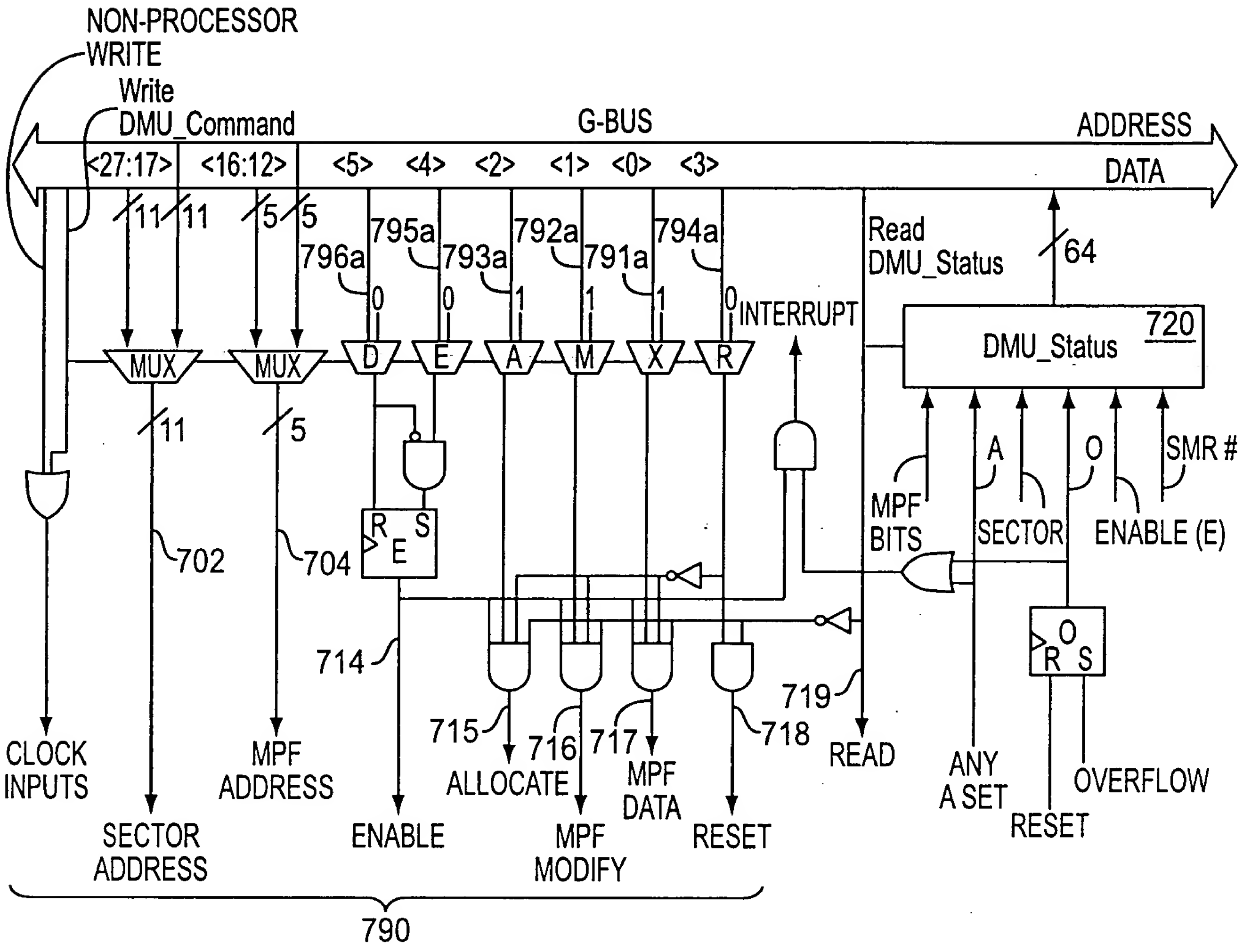

NON-PROCESSOR
WRITE
Write
DMU_Command
G-BUS
ADDRESS
DATA
<27:17>
<16:12>
<5>
<4>
<2>
<1>
<0>
<3>
11
11
5
5
796a
795a
793a
792a
791a
794a
0
0
1
1
1
0
INTERRUPT
Read
DMU_Status
64
DMU_Status
720
MUX
MUX
D
E
A
M
X
R
11
5
R
E
S
702
704
A
O
SMR #
MPF
BITS
SECTOR
ENABLE (E)
O
R
S
714
715
716
717
718
719
CLOCK
INPUTS
SECTOR
ADDRESS
MPF
ADDRESS
ENABLE
ALLOCATE
MPF
MODIFY
MPF
DATA
RESET
READ
ANY
A SET
RESET
OVERFLOW
790

FIG. 7B

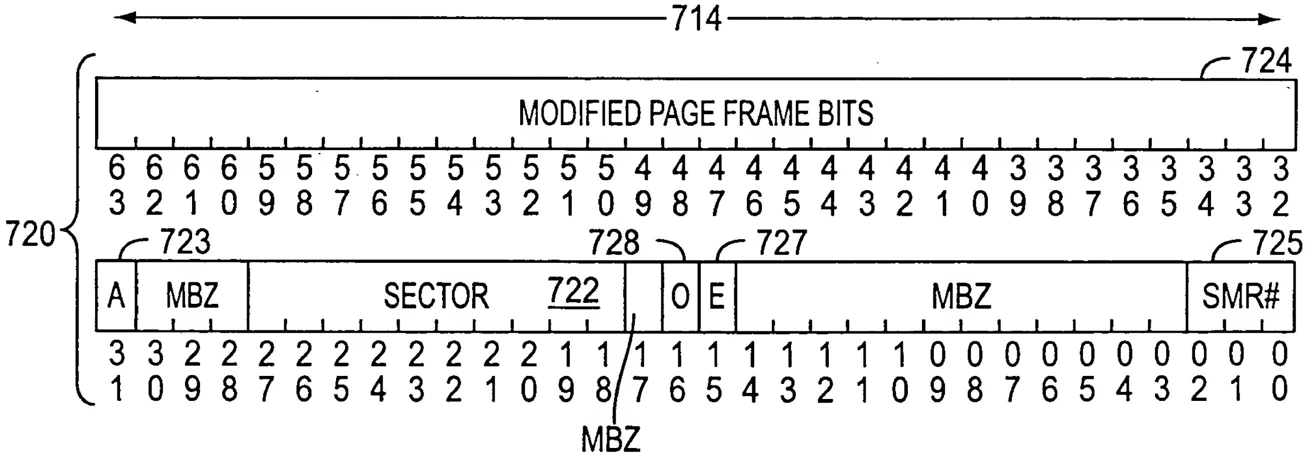

714
724
MODIFIED PAGE FRAME BITS
63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
720
723
728
727
725
A
MBZ
SECTOR
722
O
E
MBZ
SMR#
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
MBZ

FIG. 7C

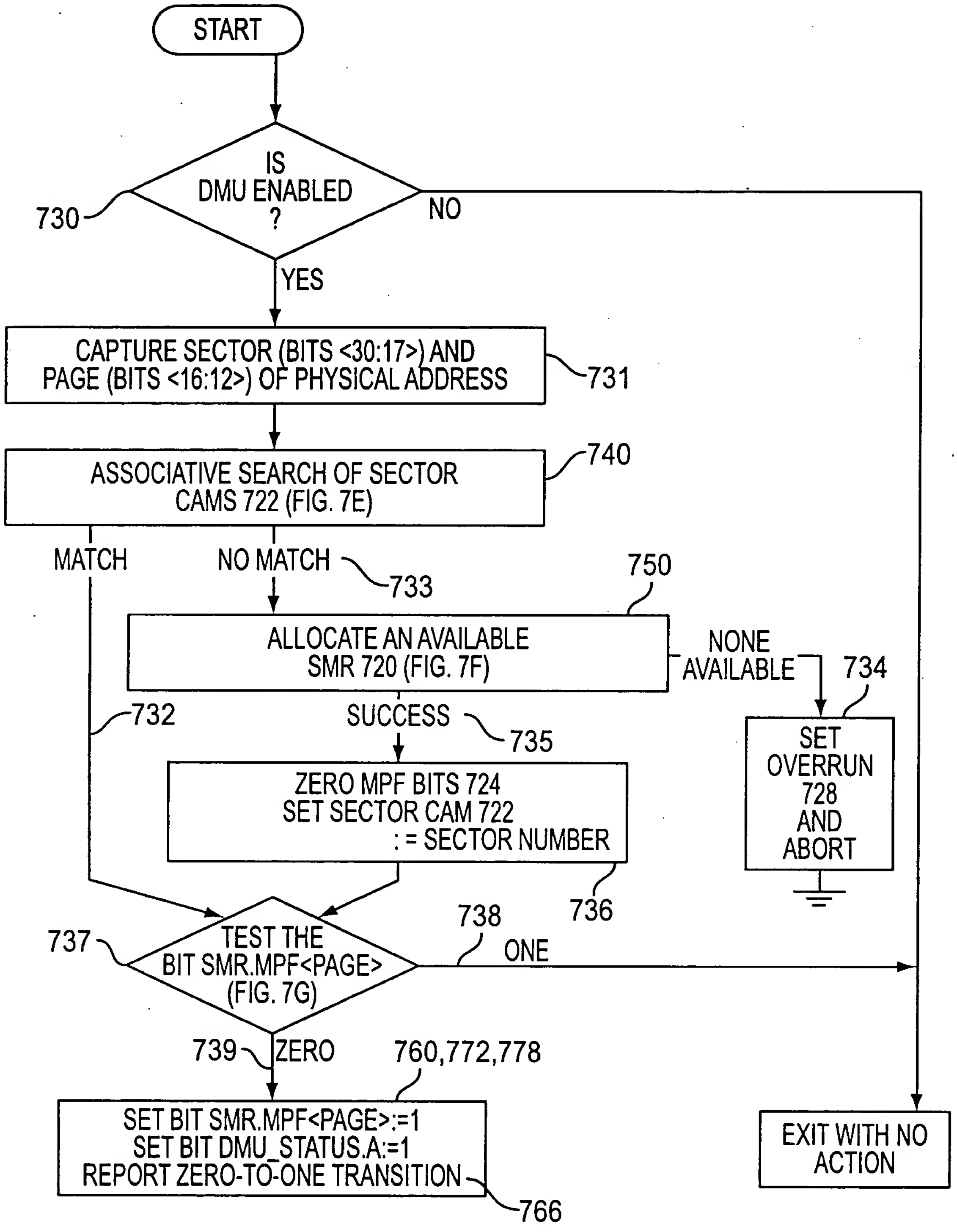
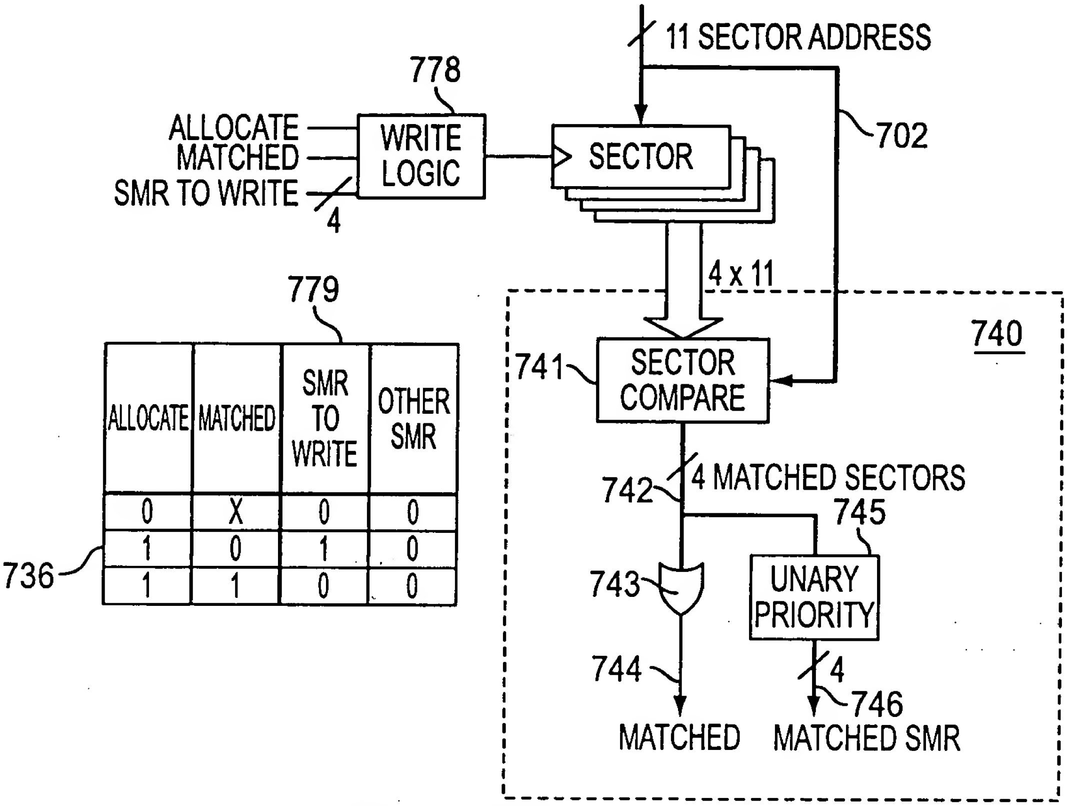

START
IS DMU ENABLED ?
730
NO
YES
CAPTURE SECTOR (BITS <30:17>) AND PAGE (BITS <16:12>) OF PHYSICAL ADDRESS
731
ASSOCIATIVE SEARCH OF SECTOR CAMS 722 (FIG. 7E)
740
MATCH
NO MATCH
733
750
ALLOCATE AN AVAILABLE SMR 720 (FIG. 7F)
NONE AVAILABLE
734
732
SUCCESS
735
SET OVERRUN 728 AND ABORT
ZERO MPF BITS 724
SET SECTOR CAM 722
: = SECTOR NUMBER
736
737
TEST THE BIT SMR.MPF<PAGE> (FIG. 7G)
738
ONE
739
ZERO
760,772,778
SET BIT SMR.MPF<PAGE>:=1
SET BIT DMU_STATUS.A:=1
REPORT ZERO-TO-ONE TRANSITION
766
EXIT WITH NO ACTION

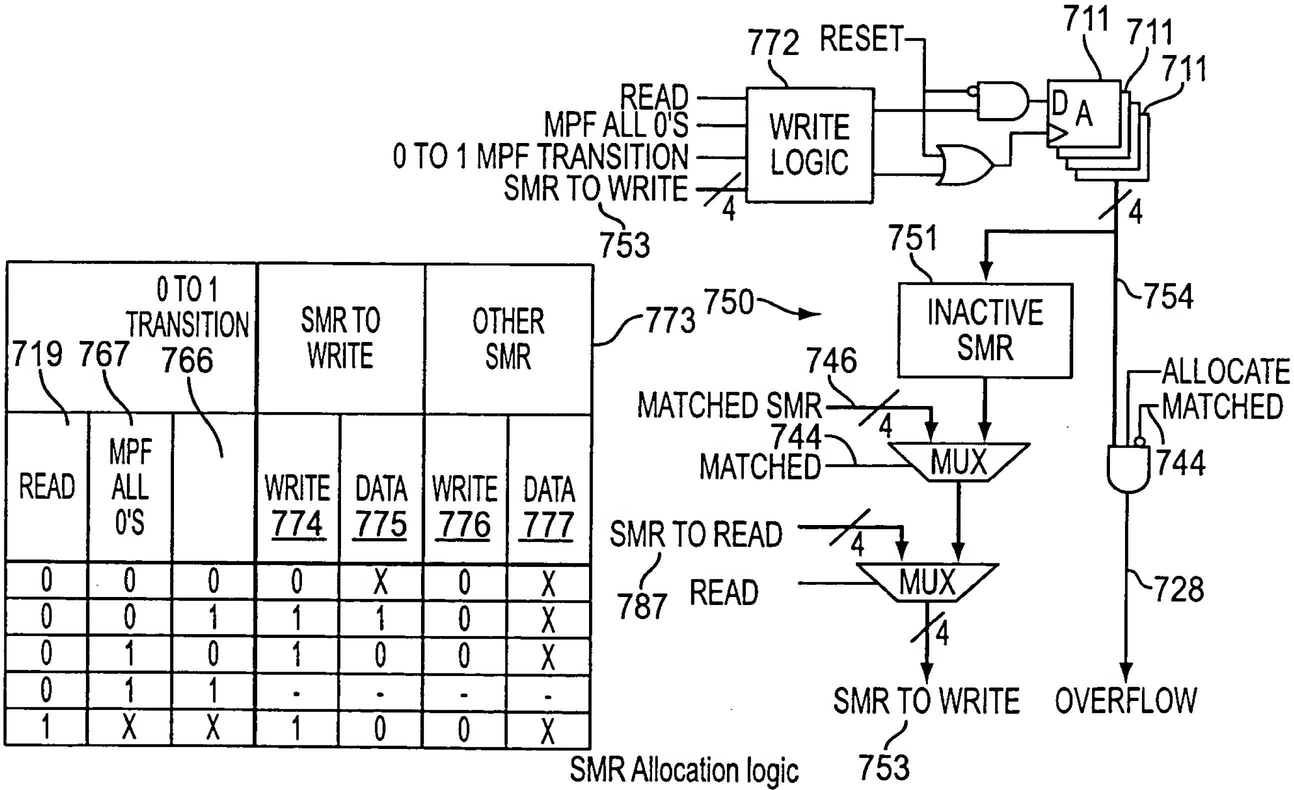
FIG. 7D



ALLOCATE	MATCHED	SMR TO WRITE	OTHER SMR
0	X	0	0
1	0	1	0
1	1	0	0

Sector match logic

FIG. 7E



719 READ	767 MPF ALL 0'S	0 TO 1 TRANSITION 766	SMR TO WRITE		OTHER SMR	
			WRITE 774	DATA 775	WRITE 776	DATA 777
0	0	0	0	X	0	X
0	0	1	1	1	0	X
0	1	0	1	0	0	X
0	1	1	-	-	-	-
1	X	X	1	0	0	X

SMR Allocation logic

FIG. 7F

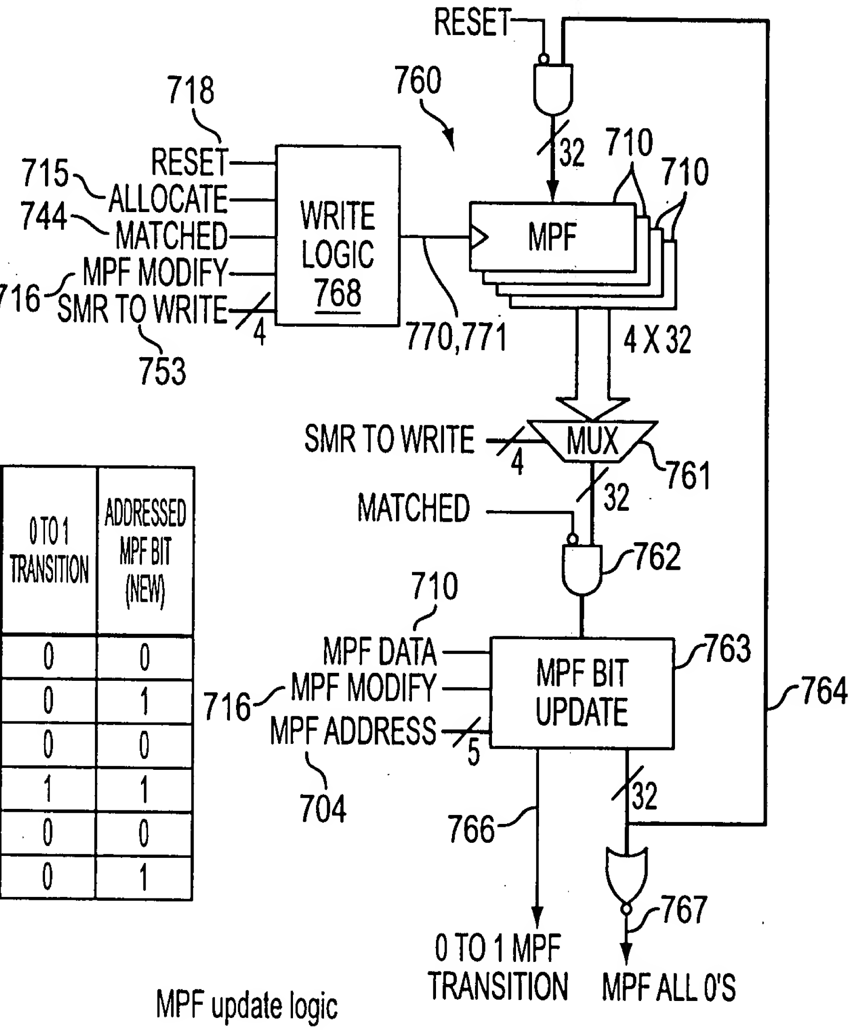
718 715 744 716 770 771

736

RESET	ALLOCATE	MATCHED	MPF MODIFY	SMR TO WRITE	OTHER SMR
0	X	X	0	0	0
0	0	0	1	0	0
0	X	1	1	1	0
0	1	X	1	1	0
1	X	X	X	1	1

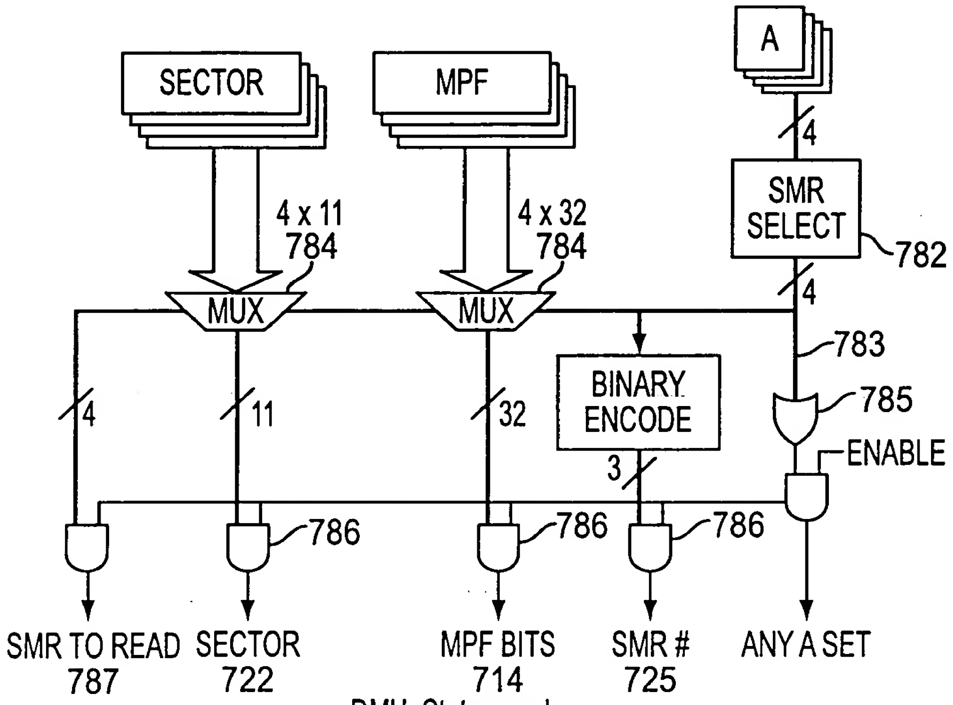
765

MPF MODIFY	ADDRESSED MPF BIT (OLD)	MPF DATA (NEW)	0 TO 1 TRANSITION	ADDRESSED MPF BIT (NEW)
0	0	X	0	0
0	1	X	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	1



MPF update logic

FIG. 7G



DMU_Status read

FIG. 7H

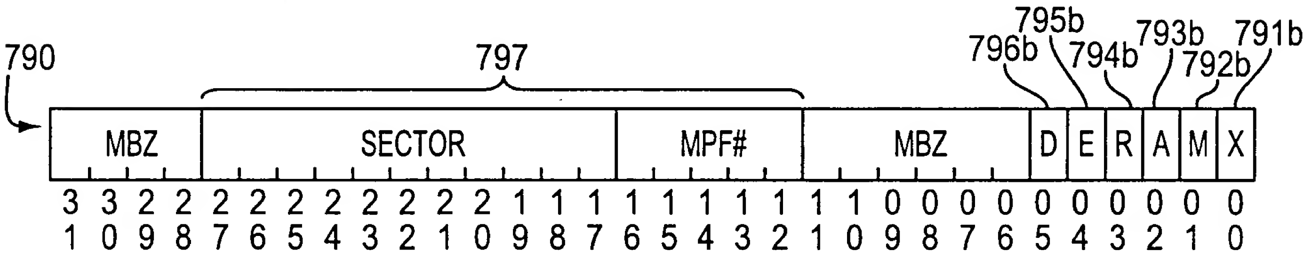
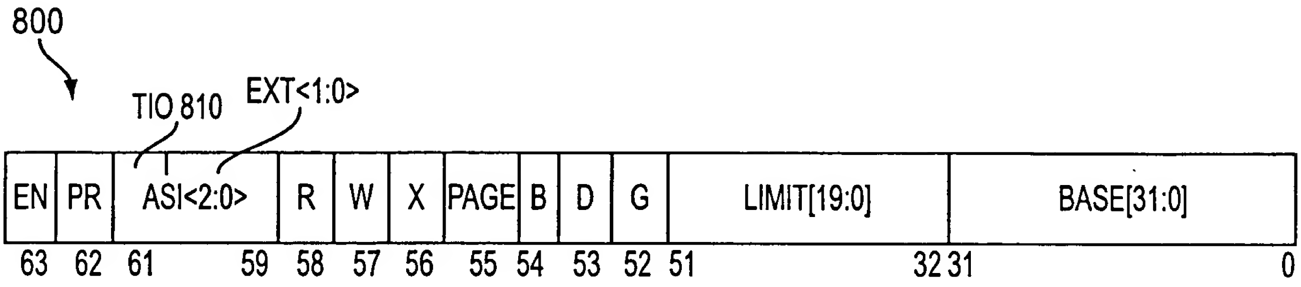


FIG. 7I

COMMAND BIT	BIT POSITION	MEANING
D	5	DISABLE MONITORING OF DMA WRITES BY CLEARING THE DMU ENABLE FLAG
E	4	ENABLE MONITORING OF DMA WRITES BY SETTING THE DMU ENABLE FLAG
R	3	RESET ALL SMRS: CLEAR ALL A AND MPF BITS AND CLEAR THE DMU OVERRUN FLAG
A	2	ALLOCATE AN INACTIVE SMR ON A FAILED SEARCH
M	1	ALLOW MPF MODIFICATIONS
X	0	NEW MPF BIT VALUE TO RECORD ON SUCCESSFUL SEARCH (OR ALLOCATION)

M	X	ACTION
0	-	INHIBIT MODIFICATION OF THE MPF BIT
1	0	CLEAR THE CORRESPONDING MPF BIT
1	1	SET THE CORRESPONDING MPF BIT

FIG. 7J



SIZE	BIT(S)	NAME	FUNCTION
1	63	SEG.EN	ENABLES SEGMENT LIMIT/PROTECTION CHECKING
1	62	SEG.PR	CHOOSES WHICH PROTECTION BITS TO USE FOR PAGE TABLE PROTECTION - (0 MEANS PSW.UK OR 1 MEANS MISC.UK)
3	61:59	SEG.AS	ADDRESS SPACE (ONLY USED WHEN SEG.PAGE IS 0)
		SEG.TIO, SEG.EXT	ADDRESS SPACE EXTENSION (ONLY USED WHEN SEG.PAGE IS 1)
3	58:56	SEG.RWX	READ/WRITE/EXECUTE '1' MEANS ENABLED - ALL 000 MEANS IT'S AN INVALID SEGMENT
1	55	SEG.PAGE	ENABLES THE PAGING SYSTEM -- (TRANSLATION AND CHECKING)
1	54	SEG.B	SEGMENT SIZE (1 MEANS 32-BIT, 0 MEANS 16-BIT)
1	53	SEG.D	SEGMENT DIRECTION (0 MEANS EXPAND UP)
1	52	SEG.G	SIZE OF LIMIT (1 MEANS IT'S IN 4k PAGES)
20	51:32	SEG.LIMIT	SEGMENT LIMIT
32	31:0	SEG.BASE	SEGMENT BASE

FIG. 8A

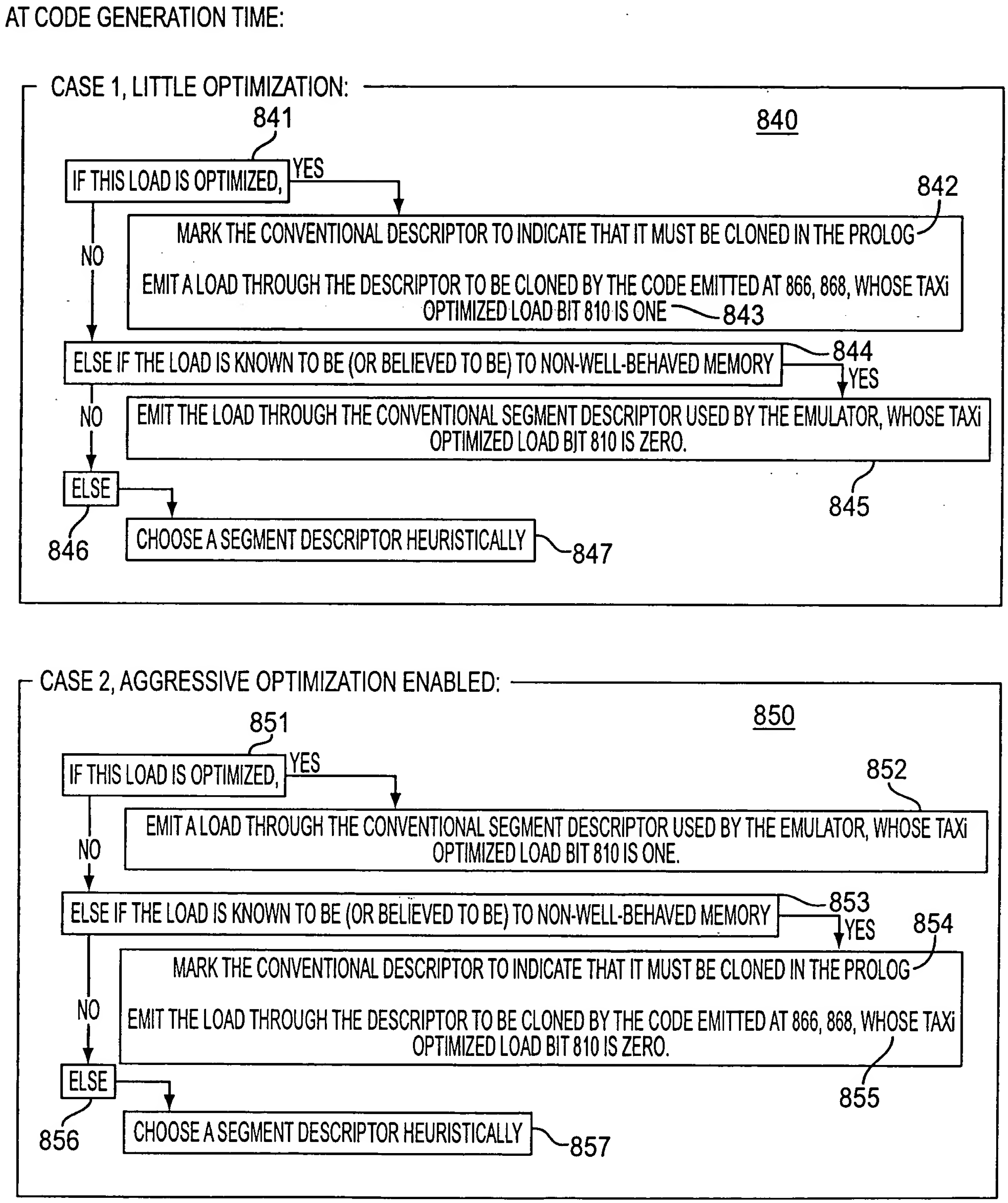

AT CODE GENERATION TIME:
CASE 1, LITTLE OPTIMIZATION:
840
841
IF THIS LOAD IS OPTIMIZED,
YES
842
MARK THE CONVENTIONAL DESCRIPTOR TO INDICATE THAT IT MUST BE CLONED IN THE PROLOG
NO
EMIT A LOAD THROUGH THE DESCRIPTOR TO BE CLONED BY THE CODE EMITTED AT 866, 868, WHOSE TAXi OPTIMIZED LOAD BIT 810 IS ONE
843
ELSE IF THE LOAD IS KNOWN TO BE (OR BELIEVED TO BE) TO NON-WELL-BEHAVED MEMORY
844
YES
NO
EMIT THE LOAD THROUGH THE CONVENTIONAL SEGMENT DESCRIPTOR USED BY THE EMULATOR, WHOSE TAXi OPTIMIZED LOAD BJT 810 IS ZERO.
845
ELSE
846
CHOOSE A SEGMENT DESCRIPTOR HEURISTICALLY
847
CASE 2, AGGRESSIVE OPTIMIZATION ENABLED:
850
851
IF THIS LOAD IS OPTIMIZED,
YES
852
EMIT A LOAD THROUGH THE CONVENTIONAL SEGMENT DESCRIPTOR USED BY THE EMULATOR, WHOSE TAXi OPTIMIZED LOAD BIT 810 IS ONE.
NO
ELSE IF THE LOAD IS KNOWN TO BE (OR BELIEVED TO BE) TO NON-WELL-BEHAVED MEMORY
853
YES
854
MARK THE CONVENTIONAL DESCRIPTOR TO INDICATE THAT IT MUST BE CLONED IN THE PROLOG
NO
EMIT THE LOAD THROUGH THE DESCRIPTOR TO BE CLONED BY THE CODE EMITTED AT 866, 868, WHOSE TAXi OPTIMIZED LOAD BIT 810 IS ZERO.
855
ELSE
856
CHOOSE A SEGMENT DESCRIPTOR HEURISTICALLY
857

FIG. 8B

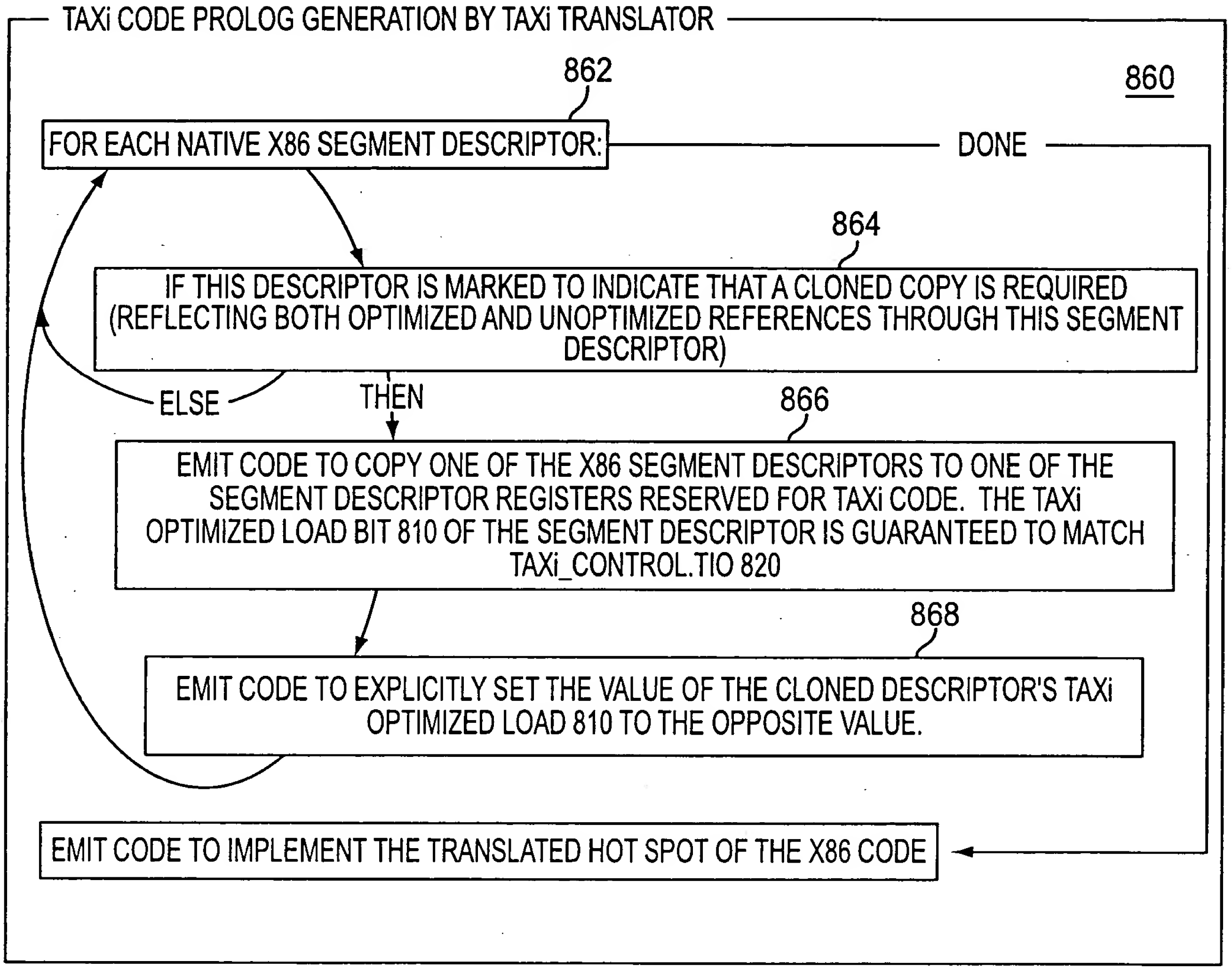

TAXi CODE PROLOG GENERATION BY TAXi TRANSLATOR
860
862
FOR EACH NATIVE X86 SEGMENT DESCRIPTOR:
DONE
864
IF THIS DESCRIPTOR IS MARKED TO INDICATE THAT A CLONED COPY IS REQUIRED (REFLECTING BOTH OPTIMIZED AND UNOPTIMIZED REFERENCES THROUGH THIS SEGMENT DESCRIPTOR)
ELSE
THEN
866
EMIT CODE TO COPY ONE OF THE X86 SEGMENT DESCRIPTORS TO ONE OF THE SEGMENT DESCRIPTOR REGISTERS RESERVED FOR TAXi CODE. THE TAXi OPTIMIZED LOAD BIT 810 OF THE SEGMENT DESCRIPTOR IS GUARANTEED TO MATCH TAXi_CONTROL.TIO 820
868
EMIT CODE TO EXPLICITLY SET THE VALUE OF THE CLONED DESCRIPTOR'S TAXi OPTIMIZED LOAD 810 TO THE OPPOSITE VALUE.
EMIT CODE TO IMPLEMENT THE TRANSLATED HOT SPOT OF THE X86 CODE

FIG. 8C